Hardware-informed reinforcement learning for quantum gate scheduling

Jakub Pietrzak[1]

Supervisor(s): Sebastian Feld[1]**, Akash Kundu**[1]**, Matthijs Spaan**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Jakub Pietrzak
Final project course: CSE3000 Research Project
Thesis committee: Matthijs Spaan, Sebastian Feld, Anna Lukina

Abstract

Quantum gate scheduling assigns start cycles to quantum-circuit operations while respecting precedence, resource, and hardware constraints. Although schedules are commonly evaluated by makespan, it is only an indirect proxy for execution reliability, since schedules of equal duration may differ in gate errors, idle-time decoherence, and crosstalk exposure. This thesis investigates whether reinforcement learning benefits from domain knowledge in quantum gate scheduling. Building on qgym's scheduling environment, we evaluate Maskable Proximal Policy Optimization against greedy ASAP and ALAP baselines on Random, GHZ, QFT, and QAOA circuit families using IBM calibration data. We study commutation-awareness, which relaxes unnecessary ordering constraints between commuting gates, and hardware-awareness, which injects calibration data through extended observations and/or a log-ESP-based reward. The main finding is that commutation-awareness is the most reliable improvement: it reduces makespan by approximately 20% for QAOA and Random circuits, while giving little benefit for GHZ and QFT circuits. Furthermore, noise-aware observation space proves promising for further research.

1 Introduction

Quantum computers offer quadratic, and in some cases exponential speedups over the best-known classical algorithms for several important computational problems, such as unstructured search [1, 2], integer factorization and discrete logarithms [2, 3], quantum random walk [2, 4]. One of the leading paradigms of quantum algorithm design is gate-based quantum hardware [5], such as superconducting qubits [6]. In this paradigm, operations on qubits can be thought of as hardware-agnostic gates that provide abstraction of hardware implementation details irrelevant to the programmer [5, 7]. A quantum program can then be represented as a sequence of gate applications used to express algorithms independently of a particular backend. However, such quantum programs must be compiled before they can be executed so that the hardware has all the necessary information for the physical realization of the gates: the initial program might have used gates unsupported by the hardware, ignored restricted qubit connectivity or used noisy paths [8, 9, 10]. Therefore, compilation is a problem at the heart of quantum computing.

Since we are in the Noisy Intermediate-Scale Quantum (NISQ) era [11], characterized by low qubit fidelity, noise, and severe hardware constraints, it is imperative to optimize the process of compilation as much as possible, yet it is a difficult combinatorial optimization problem [12]. Current work offers strong but hand-designed heuristics, which may not generalize well across different circuits, topologies, or objective functions [13, 14, 15, 16]. Particularly, ASAP/ALAP-style greedy schedulers are commonly used as baseline policies in scheduling [7, 16, 17, 18], and much of the scheduling literature formulates the objective as minimizing the makespan [7, 15, 19, 20]; however, this is only an indirect proxy for execution quality, since maximizing output fidelity or success probability requires hardware-informed objectives such as gate-dependent error modeling [21, 22].

This paper focuses on one of the compilation passes, the problem of gate scheduling. The ultimate goal of scheduling is to produce a schedule of gate executions such that the probability of an error occurring throughout the program runtime is minimized [7, 21]. In practice, a more convenient proxy metric is used for evaluation, the makespan of the schedule. The general idea is that longer circuit execution times increase exposure to decoherence and error accumulation, so shorter schedules are often expected to have higher fidelity [15, 17, 19, 20]. This is, however, not always true, since final fidelity depends on effects such as decoherence and crosstalk [21]. Furthermore, scheduling is subject to constraints, such as commutation rules and hardware limitations. It is therefore similar to the job-shop scheduling problem, which is known to be an NP-hard problem [7, 15, 23].

This decision-making problem admits a natural reinforcement-learning (RL) formulation, where an agent incrementally constructs a schedule under validity constraints. Yet, even though several attempts have been made to incorporate RL into other compilation passes [12, 24, 25, 26, 27], RL scheduling remains a relatively overlooked area. We aim to bridge the gap and analyze whether injecting more domain knowledge into the solution makes it more efficient. We will compare our method to the greedy schedulers and RL agent without domain knowledge injected.

Our methodology is chosen to answer the following questions:

Research Question 1 (RQ1): *Does commutation-awareness improve performance of schedulers or does it increase the action space size beyond their capabilities?*

Research Question 2 (RQ2): *Does awareness of real hardware error data increase the ESP of the scheduled circuits? What is the best way to inject error data-awareness?*

Research Question 3 (RQ3): *How does the Reinforcement Learning approach compare to classical schedulers in terms of makespan, ESP and inference time?*

To answer these questions, we extend qgym's scheduling setup with a conservative commutation rulebook, action masking, and a hardware-aware reliability model based on log-Estimated Success Probability (log-ESP). We compare greedy ASAP and ALAP schedulers with MaskablePPO agents on Random, GHZ, QFT, and QAOA circuit families. The experiments are divided into two benchmarks: a commutation-aware benchmark, which tests whether additional valid gate orderings improve makespan, and a noise-aware benchmark, which tests whether backend calibration data are useful when injected through observations, rewards, or both.

The results indicate that domain knowledge is useful, but only in specific forms. Commutation-awareness improves makespan substantially for QAOA and Random circuits, but has little effect on GHZ and QFT circuits. The relation between makespan and log-ESP is also circuit-dependent. Random, QAOA, and GHZ circuits show strong negative correlation between makespan and log-ESP, whereas QFT shows a much weaker correlation. However, Random circuits exhibit noticeably largest log-ESP variance among schedules of the same circuit with the same makespan. This suggests irregular circuits benefit most from hardware-informed methods.

The reinforcement-learning results are mixed. Vanilla MaskablePPO is competitive with deterministic ASAP and ALAP baselines on the tested small instances, but incurs substantially higher inference time. The noise-aware reward reduces robustness: agents trained with this reward fail significantly more often than agents trained with the standard makespan-oriented reward, especially as the circuit size increases. Adding noise data to the observation space does not harm the success rate, and remains promising for further study with larger problem instances.

In section 2, we introduce background information, problem formulation, relevant tools from the literature, and related work. Section 3 explains our methodology, the architecture of the Reinforcement Learning agent, and the experimental setup. Section 4 presents and discusses our results. Section 5 covers ethical considerations, reproducibility, replicability and responsibility of

our work. Appendix A explains all commutation rules used in this paper, Appendix B describes how we convert the gate execution times provided by IBM to abstract cycles used in our environment, Appendix C provides a derivation of the decoherence model used in this work, and Appendix D presents statistical tools we use to investigate the correlation between makespan and fidelity.

2 Background and Related Work

This section explains the problem and previous approaches in the literature. We introduce relevant tools to investigate the problem. We show related work and explain the differences with our work.

2.1 Quantum computation in near-term quantum hardware

Gate-based quantum programs are usually represented as quantum circuits: finite sequences of quantum gates applied to qubits [5]. At the algorithmic level, the qubits in a circuit are *logical qubits*; they specify the computation independently of a particular device [5]. However, before execution, a circuit must be transformed into operations that are valid for a concrete backend [8, 16]. This transformation includes decomposing the circuit into the backend's supported native operations, assigning logical qubits to physical qubits, routing two-qubit gates through the available connectivity, optimizing the resulting circuit, and scheduling the physical operations in time [8, 16].

The distinction between logical and physical qubits is especially important for Noisy Intermediate-Scale Quantum (NISQ) devices. These devices are not fault-tolerant and are affected by gate errors, readout errors, decoherence, calibration variability, and limited connectivity [11, 21, 22]. A compiled circuit can be logically equivalent to the input circuit and still perform poorly if it uses qubits with high error rates or short decoherence times [21], inserts too many additional operations during routing [13], or keeps qubits idle for too long, thereby increasing exposure to decoherence [15, 21].

In NISQ devices, connectivity is often limited: two-qubit operations can usually be applied directly only between certain pairs of physical qubits [13]. The way the physical qubits are connected to each other is called the **topology**, and is commonly represented as a coupling graph whose nodes are physical qubits and whose edges indicate permitted two-qubit interactions [16]. Qiskit's `CouplingMap`, for example, is a directed graph whose nodes are physical qubits and whose directed edges correspond to permitted CNOT interactions [28]. If a two-qubit gate has to be executed between logical qubits that are mapped to non-adjacent physical qubits, the compiler must transform the circuit, typically by inserting SWAP operations to move quantum information through the coupling graph [7, 8, 13, 28]. These additional gates increase execution time and introduce additional error opportunities.

2.2 Quantum compilation and gate scheduling

Modern quantum compilation is usually organized into several passes. Qiskit's staged transpilation pipeline, for instance, includes initialization, layout, routing, translation, optimization, and scheduling stages [28], whereas qgym considers initial mapping, routing, and scheduling environments [7]. Layout or initial mapping assigns logical qubits to physical qubits. Routing modifies the circuit so that all two-qubit operations satisfy the device connectivity. Translation rewrites gates into the target basis. Optimization reduces redundant structure, gate count, depth, or other cost measures. Scheduling then assigns execution times to operations while respecting dependencies, operation durations, and hardware resource constraints.

This work focuses on the scheduling stage. Given a circuit whose gates are already valid for the target backend, the scheduler decides when each operation should be executed. Let g_i be an operation with duration d_i and start time s_i. If g_i must precede g_j, a valid schedule must satisfy

$$s_i + d_i \leq s_j. \tag{1}$$

In addition, operations that use the same physical qubit must not overlap in time. The length of a schedule is commonly measured by its makespan,

$$C_{\max} = \max_i (s_i + d_i). \tag{2}$$

Quantum operation scheduling is often formulated as minimizing this makespan, also referred to as schedule length or overall execution time [7, 15]. This objective is physically motivated: shorter schedules reduce the time during which qubits are exposed to decoherence and therefore tend to improve the probability of obtaining a correct output [15, 21].

Quantum scheduling is related to job-shop scheduling, which is NP-hard in the classical setting [23], and although quantum scheduling is not necessarily NP-hard, it still remains a non-trivial problem [15]. Quantum instances add further structure: gate commutation, qubit-specific resource constraints, device-dependent durations, and possible execution exclusions. Commutation is particularly relevant because the textual or circuit-diagram order of gates is not always the only semantically valid order. If two gates commute, the scheduler may be able to reorder them without changing the implemented unitary, which can reduce idle time or shorten the schedule. Itoko and Imamichi formulate quantum operation scheduling as a scheduling problem with commutation and precedence constraints, showing that commutation-aware scheduling can reduce execution time [15]. In an RL formulation, this increases the number of valid action choices at a state, which can make exploration harder.

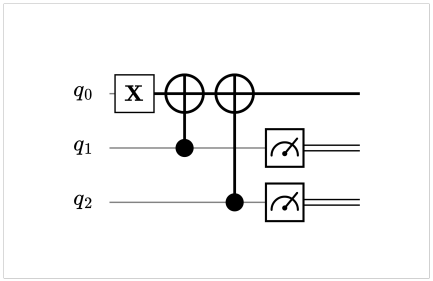
Most practical compilers rely on heuristics because mapping, routing, and scheduling are combinatorial. SABRE, for instance, uses SWAP-based bidirectional heuristic search for routing on NISQ devices with arbitrary connectivity [13]. TKET-style routing similarly targets architectures with restricted connectivity [14]. Qiskit uses simple greedy schedulers [28]. These heuristic methods are effective baselines, but their behavior is tied to manually designed cost functions.

2.3 Makespan and hardware-aware fidelity objectives

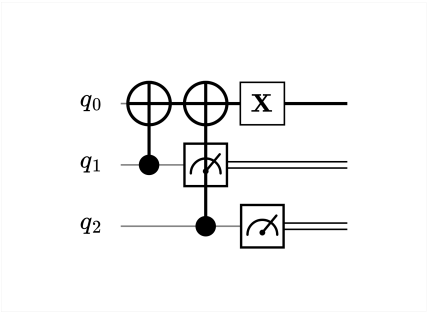
Makespan is a convenient evaluation metric, but it is blind to which physical qubits, gates, and simultaneous operations are used. Two schedules with the same makespan can therefore have different estimated reliabilities if they place operations on qubits with different calibration data, induce different idle times, or execute mutually noisy operations in parallel. Noise-adaptive compilation shows that calibration data, gate errors, readout errors, and coherence properties can materially affect program success rates [21], while crosstalk-aware scheduling shows that parallel execution may reduce fidelity even when it shortens execution time [22]. Thus, for NISQ devices, makespan should be treated as a proxy rather than the final objective.

A common alternative is to evaluate a compiled circuit by an Estimated Success Probability (ESP), which approximates the probability that all operations in the circuit succeed. In its gate-only form, ESP is modeled as the product of the fidelities of the scheduled operations [29]:

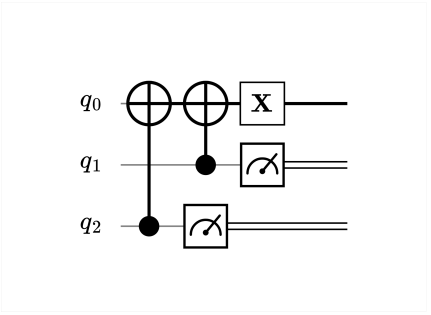
$$\mathrm{ESP}_{\mathrm{gate}} = \prod_{k=1}^{K} \prod_{g_i \in \mathcal{G}_k} F_{i,k}, \tag{3}$$



(a) The initial circuit places start cycles of the gates as respectively: $(0, 1, 2, 3, 3)$, which yields a makespan of 5.



(b) Since X gate on qubit q_0 commutes with a CNOT with a target qubit q_0, we can schedule the X gate after the CNOTs and bring measurements forward, thus getting start cycles $(0, 1, 2, 1, 2)$, and obtaining the makespan of 4. However, qubit q_2 has an idle gap of 1 cycle, which is a decoherence risk.



(c) We can commute the CNOT gates to get start cycles of $(1, 0, 2, 2, 1)$. This way the idle gap is transferred to q_1, which has a longer decoherence time. Therefore, the fidelity of the schedule has increased.

Figure 1: An example of scheduling a circuit. X and CNOT gates take 1 cycle, and measurement takes 2 cycles. Qubit q_2 has a much shorter decoherence time than q_1 and is therefore more prone to information loss when idle. Transformation from (a) to (b) shows the benefits of commutation-awareness. Transformation from (b) to (c) shows the benefits of noise-awareness with commutation-awareness. The final result of the scheduling process is the schedule with start cycles of the circuit gates of $(1, 0, 2, 2, 1)$, and the makespan of 4. The fidelity could be estimated using Eq. (7) if we knew the error data of the gates and the exact decoherence times of the qubits.

where $\mathcal{G}_k$ is the set of gates executed in time step k and $F_{i,k} \in [0, 1]$ is the estimated fidelity of gate g_i in that time step. Eq. (3) is more informative than makespan because it distinguishes operations with different error rates, but it still ignores idle-time decoherence and errors induced by simultaneous execution.

Paraskevopoulos et al. extend this idea in ArtA by including three multiplicative components: gate-operation fidelity, crosstalk penalties, and a decoherence term [30]:

$$\mathrm{ESP}_{\mathrm{ArtA}} = \left(\prod_{k=1}^{K} \prod_{g_i \in \mathcal{G}_k} F_{i,k} \right) \left(\prod_{k=1}^{K} \prod_{(i,j) \in \mathcal{X}_k} C_{i,j,k} \right) \left(e^{-t/T_2^*} \right)^N, \tag{4}$$

$$C_{i,j,k} = \left(\frac{2}{\frac{1}{F_{i,k}} + \frac{1}{F_{j,k}}} \right)^n, \tag{5}$$

where $\mathcal{X}_k$ is the set of gate pairs that incur crosstalk during time step k, $C_{i,j,k} \in [0, 1]$ is the corresponding crosstalk factor, t is the total circuit duration, T_2^* is the assumed decoherence time, N is the number of qubits, n is the number of total direct links between two gates, i.e., nearest-neighbor links, on the topology between the ith and jth operation. This formulation is useful because it makes the scheduling trade-off explicit: parallel execution can reduce t, but it can also introduce additional crosstalk penalties.

2.4 Reinforcement learning formulation

Reinforcement learning (RL) studies sequential decision-making problems in which an agent interacts with an environment and learns a policy that maximizes cumulative reward [31]. In this work, the underlying scheduling environment has a Markovian internal state $s_t \in \mathcal{S}$, which contains the information needed to update the partial schedule and compute the next reward. However, the PPO agent does not observe this full internal state directly. Instead, it receives an observation

$$o_t = \Omega(s_t),$$

where Ω is the environment's observation function and $o_t \in \mathcal{O}$, the observation space. Therefore, the scheduling problem is modeled as a finite-horizon episodic partially observable Markov decision process (POMDP), with hidden states $s \in \mathcal{S}$, observations $o \in \mathcal{O}$, actions $a \in \mathcal{A}$, transition dynamics, rewards r, observation function Ω, and discount factor $\gamma \in [0, 1]$ [32]. A policy $\pi(a \mid o)$ specifies the probability of taking action a in state s. The objective is

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T(\tau)-1} \gamma^t r_t \right].$$

where the expectation is taken over trajectories generated by following the policy [31] and each episode corresponds to one attempt to schedule a fixed circuit and terminates when all gates are scheduled or when a predefined cutoff is reached. Thus, the horizon is finite but trajectory-dependent: $T(\tau)$ denotes a terminal timestep of a trajectory τ. Gate scheduling admits a natural RL formulation because a schedule can be constructed incrementally.

The RL agent considered in this paper uses a Maskable Proximal Policy Optimization (PPO), a policy-gradient method that improves training stability by optimizing a clipped surrogate objective instead of applying unconstrained policy updates [33]. PPO is a reasonable baseline for several reasons: the original algorithm was designed to balance simplicity, sample efficiency, and stable policy updates, it is meant for a discrete, but state-dependent action space, which is the case for scheduling, and qgym already used PPO in its proof-of-concept scheduling experiment [7, 33]. We use an action mask that filters out illegal actions, such as scheduling an already scheduled gate, because it has been shown to scale better than applying the penalty for illegal action [34]. It is an important optimization, since our action space is not constant with the input size, it scales instead with $O(n)$, n being the number of gates.

2.5 Learning-based approaches to quantum compilation

Learning-based quantum compilation has received increasing attention, but most work has focused on mapping, routing, circuit optimization, or compiler-flow selection rather than hardware-aware gate scheduling. Herbert and Sengupta formulated qubit

routing as an RL problem for near-term devices [25]. Pozzi et al. later proposed a deep-Q-learning-based routing method and evaluated it against routing procedures from existing compilers [35]. Henstra further investigated how variations in RL environments and quantum-hardware characteristics affect qubit-routing performance [26]. AlphaRouter combines reinforcement learning with tree search for quantum circuit routing [27]. Other work applies deep RL to initial qubit mapping under fixed gate error rates [24], compiler-flow optimization across different passes [12], and circuit optimization using learned transformation policies [36].

These methods show that RL can improve parts of the quantum-compilation stack, but they address different decision problems. The closest prior work is qgym, which introduced RL-compatible environments for initial mapping, routing, and scheduling and demonstrated that a PPO agent can learn in a simplified scheduling setting [7].

3 Methodology

This section explains the architecture of our solution, the training setup, the evaluation, and the benchmarks.

3.1 Environment

We use `qgym`'s scheduling environment as the basis for the noise-unaware experiments [7]. The environment models quantum operation scheduling as a Markov decision problem. A circuit is represented as a list of gates, where each gate has a name and two qubit indices. For single-qubit gates these indices are equal to each other.

At timestep t, the environment state is represented as

$$s_t = (c_t,\, k_t,\, \boldsymbol{\tau}_t,\, \mathcal{M},\, \mathcal{R}_{\text{comm}},\, \mathcal{H},\, \mathcal{C}_t)\,,$$

where c_t is the current machine cycle (the point in the timeline of the schedule), k_t is the number of steps taken, $\boldsymbol{\tau}_t \in \mathbb{N}^{|\mathcal{Q}|}$ stores the remaining busy time of each qubit, $\mathcal{M}$ contains the machine properties, such as the number of qubits, the supported gate types, and gate durations in machine cycles, $\mathcal{R}_{\text{comm}}$ denotes the commutation rulebook, which defines circumstances under which a pair of gates can be reordered (see: Appendix A), and $\mathcal{C}_t$ captures circuit information such as encoded gates, dependencies, legal actions, and the partial schedule obtained so far [7].

The vector $\boldsymbol{\tau}_t$ enforces resource exclusivity through the constraint

$$a_t \in \mathcal{A}(s_t) \quad \Longrightarrow \quad \tau_t(q) = 0 \quad \forall q \in \text{supp}(a_t),$$

where $\text{supp}(a_t)$ denotes the set of qubits acted on by action a_t. This prevents overlapping operations from being scheduled on the same qubit [7].

The observation space contains gate encodings, gate dependencies, and legal actions. For the hardware-aware experiments, we extend the qgym observation and reward to include calibration data such as T_1, T_2 and gate errors: let $N_{\max}$ denote the maximum number of gate slots, N_{gate} the number of native gates in the machine's gate set, and t the current timestep, then an observation is defined as:

$$o_{\text{std},t} = (\mathbf{g},\, \mathbf{Q},\, \mathbf{D},\, \boldsymbol{\ell}\,) \in \mathcal{O}_{\text{std}}$$
$$\rightarrow o_{\text{ext},t} = \left(\mathbf{g},\, \mathbf{Q},\, \mathbf{D},\, \boldsymbol{\ell},\, \boldsymbol{\epsilon},\, \widetilde{\mathbf{T}}_1,\, \widetilde{\mathbf{T}}_2\right) \in \mathcal{O}_{\text{ext}},$$

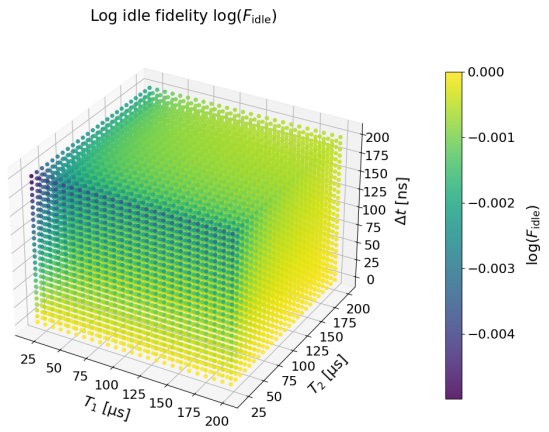


Figure 2: Visualisation of Eq. (6).

where the observation space is

$$\mathcal{O}_{\text{std}} = \{0,\ldots,N_{\text{gate}}-1\}^{N_{\max}} \times \{0,\ldots,N_q-1\}^{2N_{\max}} \times\{0,\ldots,N_{\max}-1\}^{N_{\max}} \times \{0,1\}^{N_{\max}}$$
$$\rightarrow \mathcal{O}_{\text{ext}} = \{0,\ldots,N_{\text{gate}}-1\}^{N_{\max}} \times \{0,\ldots,N_q-1\}^{2N_{\max}} \times\{0,\ldots,N_{\max}-1\}^{N_{\max}} \times \{0,1\}^{N_{\max}} \times[0,1]^{N_{\max}} \times [0,1]^{2N_{\max}} \times [0,1]^{2N_{\max}}.$$

Here, $\mathbf{g}$ encodes the gate names, $\mathbf{Q}$ stores the qubits acted on by each gate, $\mathbf{D}$ encodes the next dependency for each gate, $\boldsymbol{\ell}$ is the legal-action mask, $\boldsymbol{\epsilon}$ contains gate-error information, and $\widetilde{\mathbf{T}}_1, \widetilde{\mathbf{T}}_2$ denote the normalized T_1 and T_2 coherence times.

At each step, the agent takes an action $a \in \{0,\ldots,$ `max_gates`$\}$. If $a =$ `max_gates`, the environment increments the current machine cycle. Otherwise, the environment attempts to schedule gate a in the current cycle. Illegal actions are masked out for the Maskable PPO [34].

3.2 Estimated Success Probability

The ESP formulation in Eq. (4) contains a global decoherence factor that depends only on the total circuit duration. This form is too coarse for gate scheduling, since two schedules with the same makespan may expose different physical qubits to different idle times. Therefore, we introduce a local decoherence factor that considers qubit-specific idle-fidelity terms.

Let S be a valid schedule, $\mathcal{G}$ be its set of scheduled gates, and $\mathcal{Q}$ be the set of physical qubits. For a physical qubit q, denote by $\mathcal{I}_q(S)$ the set that contains idle intervals between two consecutive scheduled operations that act on q and an idle interval before the first operation on q. If an idle interval has duration Δt, its fidelity contribution is modeled as

$$F_{\text{idle}}(q,\Delta t) = \frac{1}{2} + \frac{2e^{-\Delta t/T_{2,q}} + e^{-\Delta t/T_{1,q}}}{6}, \tag{6}$$

where $T_{1,q}$ and $T_{2,q}$ are the relaxation and dephasing times of physical qubit q. This term is equal to 1 for $\Delta t = 0$ and decays toward $1/2$ as the idle interval grows, so longer idle gaps are penalized without forcing all qubits to share the same decoherence penalty. The theoretical motivation of the equation is explained in Appendix C. Figure 2 visualizes the equation.

Furthermore, we simplify the model to account for the fact that due to the physical difficulty of obtaining such information, we do

not possess data on gate fidelity at timestep k, but merely for the overall gate fidelity. For this reason, we drop any timestep terms from the Eq. (3) and Eq. (5).

The estimated success probability optimized in this work is then

$$\begin{aligned}\widehat{\mathrm{ESP}}(S) = &\prod_{g_i \in \mathcal{G}} F_i \\ &\cdot \prod_{(i,j)\in\mathcal{X}(S)} C_{i,j} \\ &\cdot \prod_{q\in\mathcal{Q}} \prod_{\Delta t \in \mathcal{I}_q(S)} F_{\mathrm{idle}}(q, \Delta t).\end{aligned} \tag{7}$$

where F_i is the estimated fidelity of gate g_i, $\mathcal{X}(S)$ is the set of crosstalk events induced by simultaneously scheduled gates, and $C_{i,j}$ is the crosstalk factor between gates g_i and g_j.

However, directly multiplying many probabilities can cause numerical underflow for long circuits [37]. We therefore use the logarithm of Eq. (7):

$$\begin{aligned}\mathcal{L}(S) &= \log \widehat{\mathrm{ESP}}(S) \\ &= \sum_{g_i\in\mathcal{G}} \log F_i + \sum_{(i,j)\in\mathcal{X}(S)} \log C_{i,j} \\ &\quad + \sum_{q\in\mathcal{Q}} \sum_{\Delta t\in\mathcal{I}_q(S)} \log F_{\mathrm{idle}}(q, \Delta t).\end{aligned} \tag{8}$$

This quantity is referred to as **log-ESP**. Maximizing log-fidelity is equivalent to maximizing the ESP. The two metrics are related by:

$$\widehat{\mathrm{ESP}}(S) = e^{\mathcal{L}(S)}. \tag{9}$$

The log form is also more convenient for reward computation, because multiplicative fidelity factors become additive terms that can be attributed to scheduling decisions.

3.3 Schedulers

We compare three scheduler families. The first family consists of classical baseline schedulers: ASAP [15, 16, 17, 18] and ALAP [16, 17, 18]. ASAP schedules each operation as soon as its dependencies and hardware resources allow. ALAP schedules operations as late as possible while preserving validity. If multiple gates satisfy the given greedy heuristic, the tie is decided randomly. These two policies are standard scheduling baselines and are also supported in Qiskit's scheduling pass infrastructure [28]. They are inexpensive, simple, and commonly used as reference policies in scheduling problems [7].

The second family is a vanilla reinforcement-learning scheduler based on Maskable Proximal Policy Optimization (PPO) [33, 34]. This agent receives the standard qgym scheduling observation $\mathcal{O}_{\mathrm{std}}$ and is trained with the standard reward r_{std}.

For the third family, we introduce hardware-aware PPO agents. These agents are trained with either an extended observation space $\mathcal{O}_{\mathrm{ext}}$, a noise-aware reward r_{noise}, or both. The purpose of this comparison is to isolate whether hardware information is more useful when supplied as state information, through the reward function, or through both simultaneously.

The standard reward assigns a reward of -1 per cycle advance:

$$r_{\mathrm{std},t} = \begin{cases} -1, & \text{if } a_t = \texttt{max\_gates}, \\ 0, & \text{otherwise}, \end{cases} \tag{10}$$

where a_t denotes the action taken at timestep t.

This reward primarily encourages short valid schedules and therefore targets makespan minimization.

The noise-aware reward is based on the log-fidelity objective introduced in Eq. (8). Let $\hat{S}_{t+1}$ denote the partial schedule of n gates after action a_t. The reward is computed as the log-fidelity of the current schedule normalized by the number of gates scheduled so far, with an arbitrary penalty for an empty partial schedule,

$$r_{\mathrm{noise},t} = \begin{cases} \frac{\mathcal{L}(\hat{S}_{t+1})}{n}, & \text{if } n > 0, \\ -1, & \text{if } n = 0. \end{cases} \tag{11}$$

This reward formulation allows for direct investigation of the effect of domain knowledge injection into the agent, and is motivated by the desire to optimize a direct estimation of the ultimate objective - a high-fidelity schedule. The normalization factor and the penalty ensure the agent is willing to schedule gates; scheduling a gate increases the number of gates considered by the calculation and necessarily brings down the gate fidelity term, and therefore decreases the schedule fidelity, even if the choice is optimal.

3.4 Experimental setup

Benchmarks

We evaluate two independent experiments. First, we test whether commutation-aware action spaces improve schedule quality, and whether the enlarged action space makes PPO training less sample-efficient, thus answering **RQ1**. For each scheduler family, we compare commutation-unaware and commutation-aware variants under the same observation space $\mathcal{O}_{\mathrm{std}}$, standard reward function r_{std}, training budget, and circuit set. The primary metric for this experiment is makespan. We additionally report ESP to inspect whether minimizing makespan also improves estimated execution reliability. We benchmark the performance with 3 instance size settings to investigate the agent's scalability: up to 15 gates, up to 25 gates, and up to 35 gates.

Second, we test whether hardware-aware information improves estimated execution reliability, thus answering **RQ2**. For this experiment, we use a 2×2 ablation over observation space and reward of PPO agents: standard observation $\mathcal{O}_{\mathrm{std}}$ with standard reward r_{std}, standard observation $\mathcal{O}_{\mathrm{std}}$ with noise-aware reward r_{noise}, extended observation $\mathcal{O}_{\mathrm{ext}}$ with standard reward r_{std}, and extended observation $\mathcal{O}_{\mathrm{ext}}$ with noise-aware reward r_{noise}. The primary metric is log-ESP as defined in Eq. (8). We also compare inference time across schedulers to evaluate the practical overhead of using learned policies instead of deterministic heuristics. We benchmark the performance with 3 instance size settings to investigate the agent's scalability: up to 5 gates, up to 15 gates, and up to 25 gates.

To better evaluate the importance of hardware-informed scheduling and potential sufficiency of classical schedulers (**RQ2**), we compute the correlation of makespan and log-ESP for the makespan-optimizing setup of the commutation-aware benchmark. The complete formula is introduced in the Appendix D

RQ3 is jointly answered by examining inference times in both benchmarks, makespan in the commutation-aware benchmark, and ESP in the noise-aware benchmark.

Circuits

The benchmarks use four circuit families while controlling the maximum number of gates in each circuit:

- **Random**
- **Quantum Fourier Transform (QFT)**
- **Quantum Approximate Optimization Algorithm (QAOA)**
- **Greenberger–Horne–Zeilinger (GHZ)**

The benchmark families are chosen to cover qualitatively different circuit structures while remaining consistent with benchmark choices in prior quantum-compilation work [12, 15, 27, 38].

Random circuits provide a default stress test and are commonly used to evaluate whether compilation methods can handle circuits without strong regularity [7, 27, 38].

QFT circuits represent a structured algorithmic family with long-range controlled-phase interactions and are commonly used as subroutines in quantum algorithms, such as Shor's algorithm [3, 12, 15, 27, 38].

GHZ circuits provide a simple but highly structured entangling pattern with clear dependency constraints [12, 27, 38].

QAOA circuits are widely used in compilation benchmarks because they represent variational optimization workloads with repeated cost and mixer layers, often containing many two-qubit interactions whose scheduling depends on graph structure [12, 20, 27, 38].

Together, these families span random, structured, entangling, and variational optimization-style circuits, which makes them suitable for testing whether scheduler behavior depends on circuit structure.

Hardware data

The noise model, hardware topology, and gate set are taken from IBM's FakeOslo backend, which contains seven qubits. This backend is small enough for the noise-aware RL scheduling agents to handle, while still providing sufficiently large non-uniform calibration data across qubits and gates. It therefore allows the hardware-aware schedulers to exploit differences in qubit quality, gate fidelity, and coherence times.

Additionally, we need to convert the gate set to the cycle-based model assumed by `qgym`. The conversion is described in Appendix B.

Agent training & hyperparameters

Each PPO agent is trained for 200,000 environment timesteps using `Stable-Baselines3 Contrib` implementation with the default Maskable PPO hyperparameters and `MultiInputPolicy` [39, 40]. The main PPO hyperparameters are summarized in Table 1. For each instance setting we train the agents 3 times with different seeds to ensure performance is not due to luck. Each test configuration produces 25 instances that we analyze with statistical tools. Once the agent reaches the threshold of 2500 timesteps for a particular circuit, it is assumed the agent is unable to properly schedule the circuit, and scheduling is interrupted.

Hyperparameter	Value
Policy	`MultiInputPolicy`
Learning rate	3×10^{-4}
Rollout steps n_{steps}	2048
Batch size	64
Epochs n_{epochs}	10
Discount factor γ	0.99
GAE parameter λ	0.95
Clip range	0.2
Entropy coefficient	0.0
Value-function coefficient	0.5
Maximum gradient norm	0.5

Table 1: `Stable-Baselines3 Contrib` default Maskable PPO hyperparameters used for training [40].

4 Results

In this section we discuss results of (1) commutation-aware benchmark and (2) noise-aware benchmark, both described in section 3.4. We find that commutation-awareness improves makespan for QAOA and Random circuits, vanilla MaskablePPO is competitive in both benchmarks, noise-aware reward is less robust than the standard reward, and that irregular circuits are more likely to benefit from hardware-informed methods.

4.1 Commutation-aware benchmark results

Figure 3 presents the makespan results for each circuit family and scheduler. It provides evidence for competitiveness of the RL approach against classical baselines: commutation-unaware PPO is able to perform just as well as the best commutation-unaware classical schedulers, and the performance of commutation-aware PPO is within the margin of error from the commutation-aware classical schedulers. Moreover, the figure suggests that for some circuit structures commutation-awareness may be beneficial, but not necessarily for all of them. For QAOA and Random circuits, commutation-awareness achieved $\sim 20\%$ better performance, while for GHZ and QFT the difference is negligible. Furthermore, almost all configurations show the success rate of 100%, except for one, where it is $99.11\% \pm 1.24\%$ (commutation-aware PPO; QFT circuits), which suggests that bigger problem instances could potentially be tackled by the RL approach.

Figure 4 compares inference time of the schedulers. Unsurprisingly, RL agents have a much longer inference time, with an order of magnitude difference. The shorter inference time of commutation-aware schedulers could be explained by shorter makespan and therefore smaller number of timesteps needed to conclude the scheduling.

Figure 5 presents correlation between makespan and log-ESP within same circuits. Random, QAOA, and GHZ circuits expectedly report strong negative correlation, with the coefficient r (see Eq. (56)) reported as respectively: -0.852, -0.984 and -0.661. Surprisingly, QFT reports a much weaker correlation, with $r = -0.233$, however there is a clear trend visible. Random circuits show by far largest variance standard deviation of log-ESP given makespan, with $\sigma_{y|i,m} = 0.0168$, $1-2$ orders of magnitude higher than for other families, which corresponds to approximately 1.7% relative variation in ESP, while for other circuit families, it is below 0.2%.

4.2 Noise-aware benchmark results

Figure 6 presents mean normalized log-ESP by scheduler and circuit family, while figure 7 shows the success rate of finding a valid schedule within the timestep limit; the two figures should be analyzed jointly. PPO manages to achieve a similar result as classical baselines in all circuit families except Random, where ASAP outperforms by $\sim 45\%$ in terms of log-ESP per gate. PPO with noise-aware reward manages to outperform ASAP in GHZ circuits by $\sim 14\%$ and in QFT circuits by 7% for $\mathcal{O}_{\text{ext}}$ and 32% for $\mathcal{O}_{\text{std}}$ respectively, but the results are skewed by comparatively low success rate. Whereas PPO with standard reward achieves consistently the success rate of $92.4\% - 100\%$, with noise-aware reward it reports $85.3\%/84.4\%$, $32\%/72\%$, $84.4\%/95.5\%$, and $96.9\%/96.9\%$ for GHZ, QAOA, QFT, and Random ($\mathcal{O}_{\text{std}}/\mathcal{O}_{\text{ext}}$) respectively. Overall, PPO with $\mathcal{O}_{\text{ext}}$ tends to have a better success rate. Furthermore, we find that noise-aware reward is more prone to a success rate drop as the number of gates increases.

Next, we inspect the inference time against the maximum number of gates ($n_{\max}$) in a setup, considering Random circuits. For $n_{\max} = 5$, the inference time of PPO is up to ~ 5x longer than for the classical baselines. As $n_{\max}$ grows, the factor grows: for

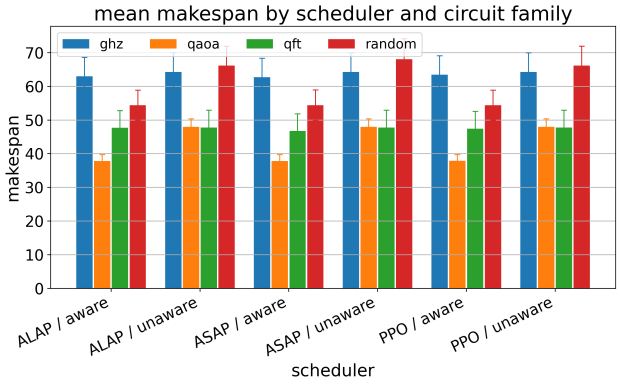


Figure 3: Commutation-aware benchmark; makespan by scheduler and circuit family. Commutation-awareness consistently shortens the mean makespan for QAOA by $\sim 21\%$ and for Random circuits by $17.7\% - 20.1\%$. For GHZ and QFT, the change is small: $1.3\% - 2.4\%$ and $0.1\% - 2.1\%$ respectively. Commutation-unaware PPO reports exactly the same performance as ALAP, which outperforms ASAP for the Random circuit family by 2.9%, but is otherwise the same. Performance of commutation-aware PPO follows closely performance of the commutation-aware classical baselines, with the differences in mean being smaller than the uncertainties.

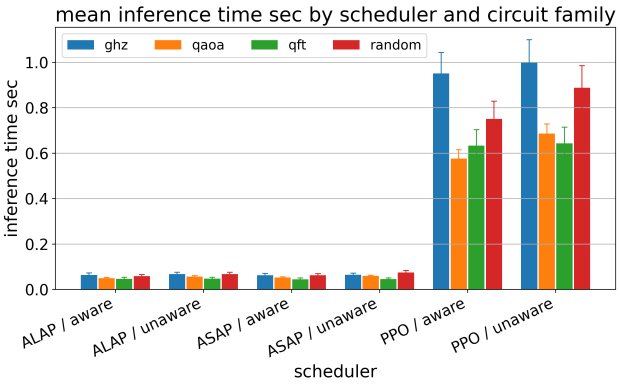


Figure 4: Commutation-aware benchmark; inference time for each circuit family and scheduler. PPO has a $\sim$ 10x $-$ 14x higher inference time than classical schedulers. Curiously, commutation-aware schedulers report shorter inference time than their commutation-unaware versions: by $2\% - 15.3\%$, $0.8\% - 12.9\%$, $1.6\% - 19.1\%$ for ALAP, ASAP, and PPO respectively.

$n_{\max} = 15$ it is $\sim$ 12x, for $n_{\max} = 25$ it is $\sim$ 14x. For $n_{\max} = 25$ PPO with extended observation space and noise-aware reward is 10% slower than PPO with standard observation space and standard reward.

4.3 Discussion

The results suggest the following answers to the research questions:

- **RQ1:** Commutation-awareness helps for some kinds of structures, but not for all of them: it shows improvements for QAOA and Random circuits, but not for GHZ and QFT. This can be explained by referring to the structures of those families. GHZ circuits are bottlenecked by repeated interactions with q_0, while QFT contains regular Hadamard barriers that limit useful reordering. QAOA and Random circuits contain more mutually commuting or parallel gates, which makes them benefit from commutation-awareness.
- **RQ2:** Hardware-aware reward hurts training robustness; standard PPO is more reliable in obtaining a valid schedule. This suggests that the reward formulation is too difficult or too sparse/noisy for the present Maskable PPO setup. The extended observation space shows a promising success rate. Noise-unaware schedulers proved to be surprisingly performant in terms of log-ESP. This suggests that perhaps considering makespan is enough, but it would be interesting to investigate performance of PPO with extended observation space and standard reward on much larger problem instances, which could potentially lead to some differences in performance. Analysis of within-circuit standard deviation of log-ESP given makespan suggests that noise-awareness should be important for irregular circuits, but that makespan may be sufficient for some highly structured circuits.
- **RQ3:** MaskablePPO can be very competitive when optimizing either makespan or log-ESP on small problem instances. It remains to be seen how it further generalizes. However, since PPO is a deep RL approach, the inference time is an order of magnitude slower than the simplest classical baselines, and the difference might grow with much larger problem sizes. A different RL algorithm may possibly overcome this in the future.

5 Responsible Research

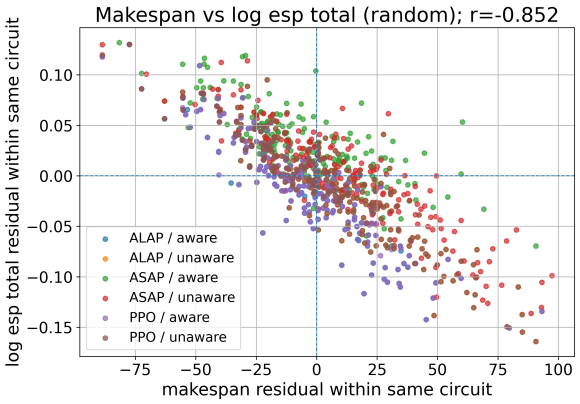
5.1 Data, reproducibility, and replicability

Our code repository is open source and available under https://github.com/GeneralKenobiJP/RLQuantumScheduler. The scripts used for obtaining the results along with the experimental data are contained within the repository to ensure replicability and reproducibility.

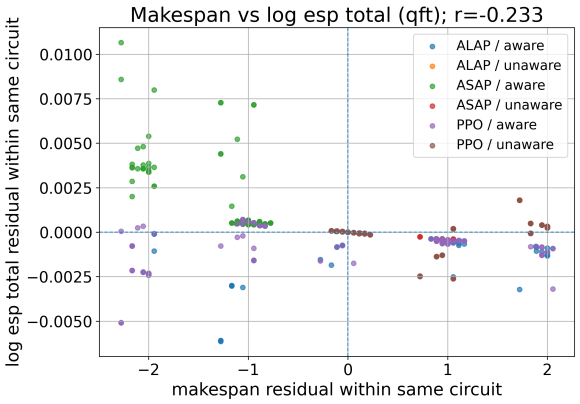
We did not use personal, sensitive, or proprietary data. Hardware information is obtained from publicly available IBM/Qiskit fake-backend calibration data, and the benchmark circuits are generated synthetically. The generated circuit sets are therefore reproducible and do not expose user data. The main reproducibility risk is not data access but stochasticity: PPO training depends on random initialization, exploration, and sampled benchmark instances. To mitigate this, the experimental scripts expose the relevant random seeds and all the settings.

5.2 Limitations

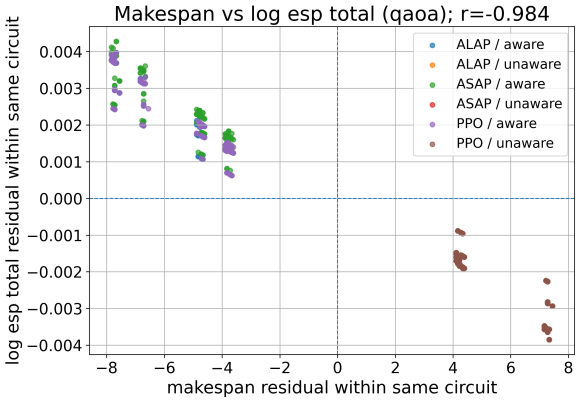
The proposed hardware-aware objective is an estimated success probability rather than a direct measurement from a real quantum processor. Consequently, the results should be interpreted as a controlled simulation study of scheduling objectives, not as a claim



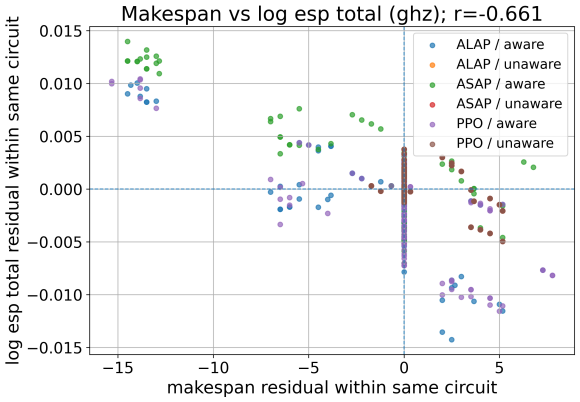
(a) Random circuits. $\sigma_{y|i,m} = 0.0168$



(b) QFT circuits. $\sigma_{y|i,m} = 0.0008$



(c) QAOA circuits. $\sigma_{y|i,m} = 0.0001$



(d) GHZ circuits. $\sigma_{y|i,m} = 0.0020$

Figure 5: Correlation of log-ESP and makespan. Each point is a scheduler result after subtracting the mean makespan and mean log-ESP of the corresponding circuit so that the schedules of different circuits are not compared. Correlation coefficient and standard deviation of log-ESP (see: Appendix D).

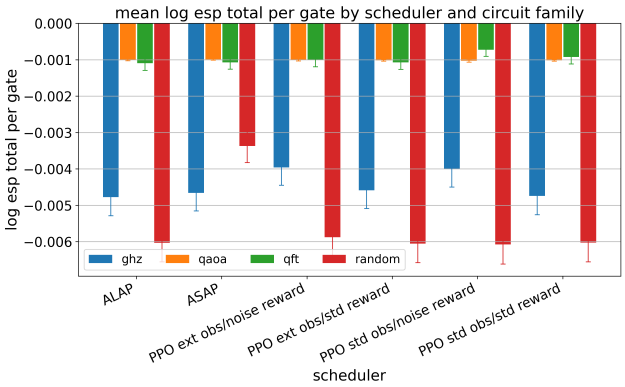


Figure 6: Noise-aware benchmark; log-ESP per gate for each circuit family and scheduler. Unsuccessful schedules are not counted. For reference, per-gate log-ESP of -0.001 results in $\sim 90.48\%$ ESP for a circuit of 100 gates, while -0.005 gives $\sim 60.65\%$.

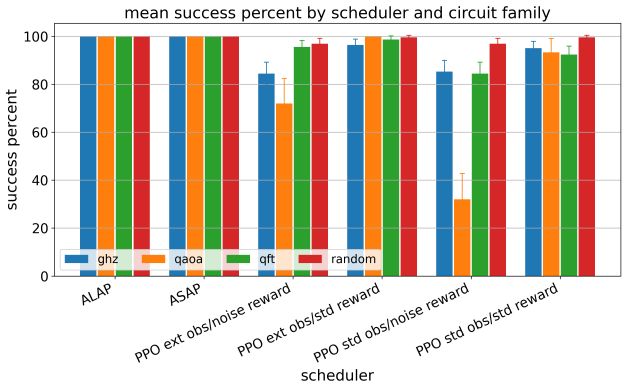


Figure 7: Noise-aware benchmark; scheduling success rate for each circuit family and scheduler. ALAP and ASAP report a 100% success rate; standard reward is more robust than noise-aware reward.

of guaranteed hardware performance. The ESP model captures gate errors, idle-time decoherence, and crosstalk penalties, but it remains an approximation and inherits the limitations of the calibration data and noise assumptions used to construct it.

The circuits considered are relatively small, which limits the ability to draw conclusions regarding generalization to larger circuits. This research should be seen as an initial result of a relatively novel approach applied to a limited scope in the context of NISQ-era computing.

5.3 Ethics

The broader motivation of this research is to improve quantum compilation, thereby helping to create larger and more complex quantum programs executable on constrained hardware. Progress in quantum compilation is relevant because quantum algorithms have potential applications in areas such as quantum simulation, chemistry, materials science, optimization, and cryptanalysis. At the same time, advances in scalable quantum computing also create security risks: sufficiently powerful quantum computers could threaten widely deployed public-key cryptographic schemes based on integer factorization and discrete logarithms. This dual-use aspect reinforces the importance of responsible development. In parallel with work on quantum computing, the research community and standardization bodies should continue the transition toward post-quantum cryptography, including recently standardized quantum-resistant algorithms.

5.4 LLM usage

Large Language Models (LLMs) were used as auxiliary tools during the project. Particularly, ChatGPT-5.5 was used to discuss ideas and the direction of research for the research plan, aid in code debugging, assist in finding references, synthesizing background knowledge, and polishing the paper. It was used to generate benchmarking code, circuit family generators, and to derive Eq. (6). Claude Sonnet 4.6 was used to check if the final approach, codebase and paper are sound. This was done for the sake of cross-examination with two different LLMs. We take full responsibility for the validity of reasoning and results presented in this work.

6 Conclusions and Future Work

Hardware-informed Reinforcement Learning (RL) is an interesting approach to the problem of quantum gate scheduling. Makespan is commonly used as the objective for schedulers, but it is only a proxy metric. Our findings suggest that even though makespan is strongly correlated with the Estimated Success Probability (ESP), a more direct schedule reliability metric, irregular circuits might benefit significantly from noise-awareness. This motivates our investigation of the effect of injecting more domain knowledge into the scheduler, using a learnable agent.

We aimed to investigate the effect of commutation-awareness and noise-awareness on the performance of the agent, and whether an RL solution is competitive against classical schedulers, such as ASAP or ALAP. To assess that, we benchmarked the schedulers against 4 circuit families: Random, GHZ, QAOA, and QFT.

We found vanilla MaskablePPO performs similarly to the baseline for all four circuit families up to 35 gates in terms of makespan and up to 25 gates in terms of ESP. Commutation-awareness improved makespan for QAOA and Random circuits, achieving $\sim$ 20% improvement in performance, but did not bring consistent improvements for GHZ or QFT circuits. The main issue with the RL solution is its inference time, which is an order of magnitude slower than for classical baselines.

We found injection of noise-awareness into the reward to be detrimental to the robustness of the PPO agent; the success rate of

PPO agents with the noise-aware reward ranges from 32% to 95.5%, while for the standard reward the success rate stays over 92.4%. The standard PPO agent and the agent with extended observation space both closely matched the classical baselines in log-ESP and success rate.

The failure of noise-aware reward may be explained by sparsity or noisiness of the noise-aware reward, or lack of extensive hyperparameter search. More research in reward formulation or reward shaping is needed to find an effective noise-aware reward or to completely reject the idea. Noise-aware observation space remains a promising topic for further investigation. Future work should focus on increasing the number of gates considered to allow for greater differentiation in the results of the main metrics.

References

[1] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[2] A. Montanaro, "Quantum algorithms: an overview," *npj Quantum Information*, vol. 2, p. 15023, 2016. [Online]. Available: https://doi.org/10.1038/npjqi.2015.23

[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997. [Online]. Available: https://doi.org/10.1137/S0097539795293172

[4] A. M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, and D. A. Spielman, "Exponential algorithmic speedup by a quantum walk," in *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 59–68. [Online]. Available: https://doi.org/10.1145/780542.780552

[5] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[6] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver, "Superconducting qubits: Current state of play," *Annual Review of Condensed Matter Physics*, vol. 11, no. Volume 11, 2020, pp. 369–395, 2020. [Online]. Available: https://www.annualreviews.org/content/journals/10.1146/annurev-conmatphys-031119-050605

[7] S. van der Linde, W. de Kok, T. Bontekoe, and S. Feld, "qgym: A gym for training and benchmarking RL-based quantum compilation," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 02. IEEE, 2023, pp. 26–30. [Online]. Available: https://arxiv.org/abs/2308.02536

[8] N. Khammassi, I. Ashraf, J. V. Someren, R. Nane, A. M. Krol, M. A. Rol, L. Lao, K. Bertels, and C. G. Almudever, "Openql: A portable quantum programming framework for quantum accelerators," *J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 1, Dec. 2021. [Online]. Available: https://doi.org/10.1145/3474222

[9] S. Khatri, R. LaRose, A. Poremba, L. Cincio, A. T. Sornborger, and P. J. Coles, "Quantum-assisted quantum compiling," *Quantum*, vol. 3, p. 140, May 2019. [Online]. Available: https://doi.org/10.22331/q-2019-05-13-140

[10] F. T. Chong, D. Franklin, and M. Martonosi, "Programming languages and compiler design for realistic quantum hardware," *Nature*, vol. 549, no. 7671, pp. 180–187, Sep. 2017.

[11] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018. [Online]. Available: https://doi.org/10.22331/q-2018-08-06-79

[12] N. Quetschlich, L. Burgholzer, and R. Wille, "Compiler optimization for quantum computing using reinforcement learning," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[13] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1001–1014. [Online]. Available: https://doi.org/10.1145/3297858.3304023

[14] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, "On the Qubit Routing Problem," in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), W. van Dam and L. Mančinska, Eds., vol. 135. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, pp. 5:1–5:32. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TQC.2019.5

[15] T. Itoko and T. Imamichi, "Scheduling of operations in quantum compiler," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2020, pp. 337–344.

[16] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with Qiskit," 2024.

[17] K. N. Smith, G. S. Ravi, P. Murali, J. M. Baker, N. Earnest, A. Javadi-Abhari, and F. T. Chong, "Error mitigation in quantum computers through instruction scheduling," 2021.

[18] L. Lao, H. van Someren, I. Ashraf, C. G. Almudever, and K. Bertels, "Mapping of lattice surgery-based quantum circuits on surface code architectures," *Quantum Science and Technology*, vol. 4, no. 1, p. 015005, 2018.

[19] K. E. C. Booth, M. Do, J. C. Beck, E. Rieffel, D. Venturelli, and J. Frank, "Comparing and integrating constraint programming and temporal planning for quantum circuit compilation," 2018.

[20] D. Venturelli, M. Do, E. Rieffel, and J. Frank, "Compiling quantum circuits to realistic hardware architectures using temporal planners," 2017.

[21] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1015–1029. [Online]. Available: https://doi.org/10.1145/3297858.3304075

[22] P. Murali, D. C. McKay, M. Martonosi, and A. Javadi-Abhari, "Software mitigation of crosstalk on noisy intermediate-scale quantum computers," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1001–1016.

[23] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 117–129, 1976.

[24] R. A. Oancea, S. van der Linde, W. de Kok, M. Sabatelli, and S. Feld, "Optimizing initial qubit mappings under fixed gate error rates using deep reinforcement learning," in *Innovations for Community Services*, S. Zielinski, G. Eichler, C. Erfurth, and G. Fahrnberger, Eds. Cham: Springer Nature Switzerland, 2025, pp. 189–208.

[25] S. Herbert and A. Sengupta, "Using reinforcement learning to find efficient qubit routing policies for deployment in near-term quantum computers," arXiv.org, 2018. [Online]. Available: https://arxiv.org/abs/1812.11619

[26] J. Henstra, "Investigating the effects of variations in reinforcement learning environments and quantum hardware for qubit routing," Master's thesis, Delft University of Technology, 02 2025. [Online]. Available: https://resolver.tudelft.nl/uuid:f33f46c5-ebea-4e2e-86e9-e6b5616134df

[27] W. Tang, Y. Duan, Y. Kharkov, R. Fakoor, E. Kessler, and Y. Shi, "Alpharouter: Quantum circuit routing with reinforcement learning and tree search," in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 01, 2024, pp. 930–940.

[28] IBM Quantum, "Qiskit Documentation," https://quantum.cloud.ibm.com/docs, 2026, accessed: 2026-05-22.

[29] N. Paraskevopoulos, F. Sebastiano, C. G. Almudever, and S. Feld, "SpinQ: Compilation strategies for scalable spin-qubit architectures," *ACM Transactions on Quantum Computing*, vol. 5, no. 1, pp. 1–36, 2023. [Online]. Available: https://doi.org/10.1145/3624484

[30] N. Paraskevopoulos, D. Hamel, A. Sarkar, C. G. Almudever, and S. Feld, "Arta: automating design space exploration of spin-qubit architectures," *Quantum Information Processing*, vol. 24, 06 2025.

[31] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[32] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1–2, pp. 99–134, 1998.

[33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[34] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," *The International FLAIRS Conference Proceedings*, vol. 35, May 2022. [Online]. Available: http://dx.doi.org/10.32473/flairs.v35i.130584

[35] 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, no. 2, pp. 1–25, 2022.

[36] T. Fösel, M. Y. Niu, F. Marquardt, and L. Li, "Quantum circuit optimization with deep reinforcement learning," arXiv.org, 2021. [Online]. Available: https://arxiv.org/abs/2103.07585

[37] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, chapter 4: Numerical Computation. [Online]. Available: https://www.deeplearningbook.org/

[38] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing," *Quantum*, vol. 7, p. 1062, Jul. 2023. [Online]. Available: https://doi.org/10.22331/q-2023-07-20-1062

[39] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, no. 1, Jan. 2021.

[40] Stable-Baselines3 Contributors, "Maskable PPO — Stable Baselines3 Contrib Documentation," https://sb3-contrib.readthedocs.io/en/master/modules/ppo_mask.html, accessed: 2026-06-12.

[41] M. A. Nielsen, "A simple formula for the average gate fidelity of a quantum dynamical operation," *Physics Letters A*, vol. 303, no. 4, pp. 249–252, 2002.

A Commutation rules

Gate scheduling is constrained not only by hardware resources, but also by the semantic order in which operations must be applied. In a circuit written as a sequence of gates, the textual order is a sufficient execution order, but it is not always necessary. If two operations commute, their order can be exchanged without changing the implemented unitary. Exposing such alternative orders can give the scheduler additional freedom and may reduce idle time or makespan. In this work, commutation is therefore used as a conservative mechanism for relaxing precedence constraints between gates.

Let two gates be denoted by g_i and g_j. Each controlled two-qubit gate is written as acting on a control qubit c and a target qubit t. For the gate representation used in the implementation, q_1 denotes the control qubit and q_2 denotes the target qubit for controlled gates. For single-qubit gates, q_1 denotes the acted-on qubit. The commutation rules used in this work are sufficient rules: whenever a rule returns true, the two gates are treated as commuting, but the rule set is not intended to decide all possible cases of unitary commutation.

The first rule concerns rotations about the same Pauli axis. Single-qubit rotations can be written as

$$R_P(\theta) = \exp\left(-i\frac{\theta}{2}P\right), \qquad P \in \{X, Y, Z\}. \tag{12}$$

Rotations generated by the same Pauli operator commute, because they are functions of the same matrix:

$$R_P(\theta_1)R_P(\theta_2) = R_P(\theta_1 + \theta_2) = R_P(\theta_2)R_P(\theta_1). \tag{13}$$

The implementation applies this rule to gates with the same rotation-axis name, including `rx`, `ry`, `rz`, their Pauli special cases `x`, `y`, `z`, and the two-qubit Pauli rotations `rxx`, `ryy`, and `rzz`.

The second rule handles pairs of CNOT gates. A CNOT gate with control c and target t may be written as

$$\mathrm{CX}_{c,t} = |0\rangle\langle 0|_c \otimes I_t + |1\rangle\langle 1|_c \otimes X_t. \tag{14}$$

Two CNOT gates are treated as commuting when the control of neither gate is the target of the other:

$$c_i \neq t_j \quad \text{and} \quad t_i \neq c_j. \tag{15}$$

This includes the common cases of CNOT gates with the same control and different targets, CNOT gates with different controls and the same target, and CNOT gates acting on disjoint qubits. The excluded case is the dependency pattern in which the target of one CNOT is the control of the other, since such gates generally do not commute.

The third rule extends the same-control case to controlled gates beyond CNOT. For two controlled gates with the same control qubit and different target qubits,

$$\Lambda(U)_{c,t_i} \quad \text{and} \quad \Lambda(V)_{c,t_j}, \qquad t_i \neq t_j, \tag{16}$$

the gates commute because they are block-diagonal with respect to the same control qubit and apply their target operations on different subsystems. This rule is used for controlled gates such as `cx`, `cz`, and `cp`.

The fourth rule concerns diagonal gates. Any two diagonal matrices commute. Therefore, gates that are diagonal in the computational basis are treated as mutually commuting. In the implementation this class includes `z`, `s`, `t`, `rz`, `cz`, `cp`, `id`, and `rzz`. This rule is useful because many phase-type operations can be reordered without changing the circuit unitary.

The fifth rule captures the interaction between R_z rotations and CNOT gates. A Z-axis rotation on the control qubit of a CNOT commutes with that CNOT:

$$R_z^{(c)}(\theta)\,\mathrm{CX}_{c,t} = \mathrm{CX}_{c,t}\,R_z^{(c)}(\theta). \tag{17}$$

This follows from the fact that the CNOT is diagonal with respect to the control-qubit projectors $|0\rangle\langle 0|$ and $|1\rangle\langle 1|$, while R_z is also diagonal in the same basis.

The sixth rule captures the dual case for R_x rotations on the target of a CNOT. Since

$$R_x(\theta) = \exp\left(-i\frac{\theta}{2}X\right), \tag{18}$$

R_x is a function of the same Pauli operator X that appears in the target action of the CNOT. Consequently,

$$R_x^{(t)}(\theta)\,\mathrm{CX}_{c,t} = \mathrm{CX}_{c,t}\,R_x^{(t)}(\theta). \tag{19}$$

Together, these rules provide a lightweight commutation rulebook for the scheduler. They expose several important and efficiently checkable reorderings, while avoiding the cost of a full symbolic or matrix-based commutation test for arbitrary gates.

B Real-time-to-cycles conversion model

Given a gate set $\mathcal{G}$ taken from the hardware data, for each gate we average out the reported mean per-qubit durations of execution and convert it to abstract cycles. The first part is needed to conform to the `qgym`'s API. The second part is performed because the runtime scales with the number of cycles; therefore, it would be inefficient to use 1 cycle per, for instance, 1ns. Instead, a cycle length should reflect the minimal notable difference between the duration of any 2 gates. Furthermore, `qgym` does not support virtual gates that take no physical time to execute. To account for that, the following conversion model is introduced: let ϵ be a precision threshold, c_i the duration of gate i in cycles, d_i the mean duration of gate i across all qubits or pairs of qubits in nanoseconds:

$$\boxed{c_i = \max\left(1, \left\lfloor \frac{d_i}{\min_{\substack{0\le k,l<|\mathcal{G}| \\ k\neq l, d_k\neq d_l}} \max\left(\epsilon, |d_k - d_l|\right)} \right\rfloor\right)} \tag{20}$$

We set $\epsilon = 1\mathrm{ns}$.

C Derivation of the Local Decoherence model

This appendix derives the idle-gap fidelity used for local decoherence modeling. Consider a physical qubit q that remains idle for a duration Δt. During this interval, the ideal operation is the identity channel, while the physical evolution is modeled as a single-qubit thermal relaxation channel parameterized by $T_{1,q}$ and $T_{2,q}$. The goal is to estimate the average qubit state fidelity under time evolution of the state between the operations on the qubit.

We use the standard average gate fidelity definition, namely the Haar average of $\langle\psi|\mathcal{E}(|\psi\rangle\langle\psi|)|\psi\rangle$ over pure input states [41].

C.1 Bloch-vector representation

An arbitrary single-qubit density matrix can be written as

$$\rho = \frac{1}{2}\left(I + xX + yY + zZ\right), \tag{21}$$

where

$$\mathbf{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{22}$$

is the Bloch vector. For pure states, $\|\mathbf{r}\| = 1$. The z-coordinate measures population imbalance $z = \rho_{00} - \rho_{11}$, the state $|0\rangle$ has $z = +1$, while $|1\rangle$ has $z = -1$. The maximally mixed state has $z = 0$.

We model relaxation phenomenologically by the standard Bloch-vector decay laws: transverse components decay with characteristic time T_2, while the longitudinal component relaxes with characteristic time T_1. This single-qubit affine-channel description is consistent with the density-matrix and quantum-operation formalism of Nielsen and Chuang [5].

Thermal relaxation does not merely shrink the Bloch sphere, but also pulls states towards the thermal equilibrium state. That pull is a translation of the Bloch sphere along the z-axis:

$$\mathbf{r} \mapsto A\mathbf{r} + c, \tag{23}$$

where

$$c = (0, 0, c_z)^T \tag{24}$$

. The affine displacement c_z depends on the thermal equilibrium polarization. For zero-temperature relaxation, the equilibrium state is $|0\rangle$, therefore $z_{\mathrm{eq}} = +1$. The longitudinal Bloch coordinate obeys:

$$z(t) = z_{\mathrm{eq}} + (z(0) - z_{\mathrm{eq}})e^{-t/T_1}. \tag{25}$$

With $z_{\mathrm{eq}} = 1$,

$$z(t) = 1 + (z(0) - 1)e^{-t/T_1} = e^{-t/T_1}z(0) + 1 - e^{-t/T_1}. \tag{26}$$

Therefore,

$$c_z = 1 - e^{-\Delta t/T_{1,q}}. \tag{27}$$

More generally, c_z depends on the excited-state equilibrium population. However, this affine displacement does not contribute to the Haar-averaged fidelity derived below.

The channel can be written as an affine map because every trace-preserving single-qubit quantum channel acts affinely on Bloch vectors. Indeed, for a linear trace-preserving channel $\mathcal{E}$,

$$\mathcal{E}(\rho) = \frac{1}{2}\mathcal{E}(I) + \frac{1}{2}\sum_{j\in x,y,z} r_j\mathcal{E}(\sigma_j),$$

and the output Bloch coordinates $r_i' = \mathrm{Tr}[\sigma_i\mathcal{E}(\rho)]$ therefore have the form

$$r_i' = c_i + \sum_j Aij r_j, \qquad c_i = \frac{1}{2}\mathrm{Tr}[\sigma_i\mathcal{E}(I)], \qquad Aij = \frac{1}{2}\mathrm{Tr}[\sigma_i\mathcal{E}(\sigma_j)].$$

Thus $\mathbf{r}' = A\mathbf{r} + \mathbf{c}$. In the thermal-relaxation model used here, the x- and y-components decay with T_2, while the z-component decays with T_1 and is shifted by the affine displacement described above.

The channel can be written as an affine map

$$\mathbf{r} \mapsto A_q(\Delta t)\mathbf{r} + \mathbf{c}, \tag{28}$$

where

$$A_q(\Delta t) = \begin{pmatrix} e^{-\Delta t/T_{2,q}} & 0 & 0 \\ 0 & e^{-\Delta t/T_{2,q}} & 0 \\ 0 & 0 & e^{-\Delta t/T_{1,q}} \end{pmatrix}. \tag{29}$$

C.2 State fidelity for a fixed input state

For two single-qubit states

$$\rho(\mathbf{r}) = \frac{1}{2}\left(I + \mathbf{r}\cdot\boldsymbol{\sigma}\right) \tag{30}$$

and

$$\rho(\mathbf{s}) = \frac{1}{2}\left(I + \mathbf{s}\cdot\boldsymbol{\sigma}\right), \tag{31}$$

where

$$\boldsymbol{\sigma} = (X, Y, Z), \tag{32}$$

their overlap is

$$\mathrm{Tr}\left[\rho(\mathbf{r})\rho(\mathbf{s})\right] = \frac{1}{2}\left(1 + \mathbf{r}\cdot\mathbf{s}\right). \tag{33}$$

For a pure input state, this overlap is the fidelity between the input state and the output state. The ideal idle operation leaves the input state unchanged, while the noisy idle output has Bloch vector

$$\mathbf{s} = A_q(\Delta t)\mathbf{r} + \mathbf{c}. \tag{34}$$

Thus,

$$F(\mathbf{r}) = \frac{1}{2}\left(1 + \mathbf{r}^T\left[A_q(\Delta t)\mathbf{r} + \mathbf{c}\right]\right) \tag{35}$$

$$= \frac{1}{2}\left(1 + \mathbf{r}^T A_q(\Delta t)\mathbf{r} + \mathbf{r}^T\mathbf{c}\right). \tag{36}$$

C.3 Haar average over input states

The average idle fidelity is obtained by averaging over all pure single-qubit input states:

$$F_{\mathrm{idle}}(q, \Delta t) = \mathbb{E}_{\mathbf{r}}\left[F(\mathbf{r})\right]. \tag{37}$$

Let $\alpha, \beta \in \{x, y, z\}$ denote axis indices of the Bloch sphere. For a uniformly distributed pure qubit state on the Bloch sphere,

$$\mathbb{E}[r_\alpha] = 0, \tag{38}$$

and

$$\mathbb{E}[r_\alpha r_\beta] = \frac{\delta_{\alpha\beta}}{3}. \tag{39}$$

The first identity follows from symmetry: for every point $\mathbf{r}$ on the Bloch sphere, the opposite point $-\mathbf{r}$ is equally likely, so the average component in every direction vanishes. For the second identity, define the second-moment matrix

$$M_{\alpha\beta} = \mathbb{E}[r_\alpha r_\beta]. \tag{40}$$

The uniform distribution on the Bloch sphere is rotationally invariant, hence M must be proportional to the identity, $M = \lambda I$. Since pure states satisfy $|\mathbf{r}|^2 = 1$,

$$1 = \mathbb{E}[|\mathbf{r}|^2] = \sum_\alpha \mathbb{E}[r_\alpha^2] = \mathrm{Tr}[M] = 3\lambda. \tag{41}$$

Thus $\lambda = 1/3$, and consequently

$$M_{\alpha\beta} = \frac{\delta_{\alpha\beta}}{3}. \tag{42}$$

Because Bloch-vector components are real, no complex conjugation is involved.

Plugging Eq. (36) into Eq. (37), we obtain:

$$F_{\mathrm{idle}}(q, \Delta t) = \frac{1}{2}\left(1 + \mathbb{E}_{\mathbf{r}}\left[\mathbf{r}^T A_q(\Delta t)\mathbf{r}\right] + \mathbb{E}_{\mathbf{r}}\left[\mathbf{r}^T\mathbf{c}\right]\right). \tag{43}$$

The affine contribution vanishes because

$$\mathbb{E}_{\mathbf{r}}\left[\mathbf{r}^T\mathbf{c}\right] = \mathbb{E}_{\mathbf{r}}[\mathbf{r}]^T\mathbf{c} = 0, \tag{44}$$

where the last equality comes from Eq. (38).

For the quadratic term,

$$\mathbb{E}_{\mathbf{r}}\left[\mathbf{r}^T A_q(\Delta t)\mathbf{r}\right] = \sum_{\alpha,\beta} A_{\alpha\beta}\mathbb{E}[r_\alpha r_\beta] \tag{45}$$

$$= \frac{1}{3}\sum_\alpha A_{\alpha\alpha} \tag{46}$$

$$= \frac{1}{3}\mathrm{Tr}\left[A_q(\Delta t)\right], \tag{47}$$

where the second equality comes from Eq. (39).

Hence,

$$F_{\mathrm{idle}}(q, \Delta t) = \frac{1}{2} + \frac{1}{6}\mathrm{Tr}\left[A_q(\Delta t)\right]. \tag{48}$$

Since

$$\mathrm{Tr}\left[A_q(\Delta t)\right] = 2e^{-\Delta t/T_{2,q}} + e^{-\Delta t/T_{1,q}}, \tag{49}$$

which we obtain by taking the trace of the matrix presented in Eq. (29), we conclude that:

$$\boxed{F_{\mathrm{idle}}(q, \Delta t) = \frac{1}{2} + \frac{2e^{-\Delta t/T_{2,q}} + e^{-\Delta t/T_{1,q}}}{6}.} \tag{50}$$

The corresponding idle error probability is

$$\boxed{p_{\mathrm{idle}}(q, \Delta t) = 1 - F_{\mathrm{idle}}(q, \Delta t).} \tag{51}$$

C.4 Physical constraint

For a Markovian single-qubit thermal relaxation channel, the relaxation parameters must satisfy

$$T_{2,q} \le 2T_{1,q}. \tag{52}$$

to retain Markovianity. Accordingly, if backend calibration data reports $T_{2,q} > 2T_{1,q}$, we use the clipped value

$$T_{2,q}^{\mathrm{eff}} = \min\left(T_{2,q}, 2T_{1,q}\right), \tag{53}$$

before evaluating Eq. (50). This mirrors the physical validity condition imposed by standard thermal-relaxation noise models.

D Makespan–fidelity correlation formula

To analyze the relationship between schedule length and fidelity, we report a within-circuit correlation rather than a correlation over all benchmark rows, which would compare circuits of different sizes and using different gates and thus provide an unfair comparison. For each circuit i of mean makespan $\bar{x}_i$ and mean log-ESP $\bar{y}_i$, and for each scheduler result j reporting makespan $x_{i,j}$ and log-ESP $y_{i,j}$, we compute:

$$\tilde{x}_{i,j} = x_{i,j} - \bar{x}_i, \tag{54}$$

$$\tilde{y}_{i,j} = y_{i,j} - \bar{y}_i \tag{55}$$

Then, the overall correlation r is computed as:

$$\boxed{r = \frac{\sum_{i=1}^{C} \sum_{j=1}^{n_i} (x_{ij} - \bar{x}_i)(y_{ij} - \bar{y}_i)}{\sqrt{\sum_{i=1}^{C} \sum_{j=1}^{n_i} (x_{ij} - \bar{x}_i)^2} \sqrt{\sum_{i=1}^{C} \sum_{j=1}^{n_i} (y_{ij} - \bar{y}_i)^2}},} \tag{56}$$

where C is the number of unique circuits, and n_i is the number of successful schedules obtained for the circuit i.

The sample within-circuit standard deviation estimate of log-ESP given makespan m is computed as:

$$\boxed{\sigma_{y|i,m} = \sqrt{\frac{\sum_{i,m} \sum_{j=1}^{n_{im}} (y_{imj} - \bar{y}_{im})^2}{\sum_{i,m} (n_{im} - 1)}},} \tag{57}$$

where n_{im} is the number of schedules of makespan m obtained for the circuit i.