Reinforcement Learning Based Quantum Circuit Optimization via ZX-Calculus

Jordi Riu[1,2], Jan Nogué[1,2], Gerard Vilaplana[1], Artur Garcia-Saez[1,3], and Marta P. Estarellas[1]

[1]Qilimanjaro Quantum Tech, Carrer de Veneçuela, 74, Sant Martí, 08019, Barcelona, Spain
[2]Universitat Politècnica de Catalunya, Carrer de Jordi Girona, 3, 08034 Barcelona, Spain
[3]Barcelona Supercomputing Center, Plaça Eusebi Güell, 1-3, 08034 Barcelona, Spain

We propose a novel Reinforcement Learning (RL) method for optimizing quantum circuits using graph-theoretic simplification rules of ZX-diagrams. The agent, trained using the Proximal Policy Optimization (PPO) algorithm, employs Graph Neural Networks to approximate the policy and value functions. We demonstrate the capacity of our approach by comparing it against the best performing ZX-Calculus-based algorithm for the problem in hand. After training on small Clifford+T circuits of 5-qubits and few tenths of gates, the agent consistently improves the state-of-the-art for this type of circuits, for at least up to 80-qubit and 2100 gates, whilst remaining competitive in terms of computational performance. Additionally, we illustrate the versatility of the agent by incorporating additional optimization routines on the workflow during training, improving the two-qubit gate count state-of-the-art on multiple structured quantum circuits for relevant applications of much larger dimension and different gate distributions than the circuits the agent trains on. This conveys the potential of tailoring the reward function to the specific characteristics of each application and hardware backend. Our approach is a valuable tool for the implementation of quantum algorithms in the near-term intermediate-scale range (NISQ).

Jordi Riu: jordi.riu@qilimanjaro.tech
Jan Nogué: jan.nogue@qilimanjaro.tech

1 Introduction

Quantum computation is a promising paradigm that exploits the principles of quantum mechanics to perform tasks that are intractable for classical computers. However, current quantum devices face significant challenges, such as the presence of noise and decoherence in the physical systems that implement quantum circuits [1]. These challenges limit the scalability and the reliability of quantum computation, and pose a major obstacle for achieving quantum advantage over classical computation. Therefore, it is essential to design and optimize quantum circuits in a way that minimizes the number of gates and the resources required, while preserving the functionality and the fidelity of the computation [2].

One of the common approaches for optimizing quantum circuits is to apply algebraic identities to perform gate permutations and gate cancellations in the original circuit [3, 4]. Using reinforcement learning (RL) [5] in combination with this approach is currently being explored with promising results [6, 7]. However, in the previous context, the action space for the RL agent grows quickly, as there are several types of gate identities that need to be identified and each of those may involve multiple gates. This makes it harder for reinforcement learning agents to explore and exploit the optimal actions, as they have to deal with a large and diverse set of possible gate permutations and cancellations.

To overcome these limitations, we instead use ZX-Calculus [8], a graphical language to reason about quantum computation, as our framework for quantum circuit optimization. Using ZX-Calculus for quantum circuit optimization has the advantage of requiring a smaller and simpler action space, as there are fewer types of rules

that can be applied to ZX-diagrams. ZX rules describe graphical ways to operate between its basic elements, spiders and wires, while preserving the semantics of the computation. Moreover, any ZX-Calculus rule can be described with at most two of these spiders.

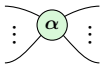
In this work, we incorporate RL and, more specifically, the Proximal Policy Optimization algorithm (PPO) [9] to guide the optimization of quantum circuits through the ZX formalism. We define a reward function that reflects the quality of the circuit optimization and explore the space of possible transformations using the ZX-Calculus rules. This paper is the result of our previous exploratory work [10], where a similar approach was applied to optimize quantum circuits with Clifford gates, which are a subset of quantum gates that can be efficiently simulated on classical computers [11, 12]. In [10], convolutional neural networks were used to learn the ZX-Calculus rewrite rules, and the method was shown to improve the existing ZX-Calculus based optimization algorithms implemented in the PyZX [13] package for small circuits. However, some limitations concerning the use of convolutional neural networks were identified, such as the difficulty in handling inputs and outputs of variable sizes, which severely limits the scalability of the approach. In this work, we improve the approach by using graph neural networks [14] instead of convolutional neural networks, as they are better suited to capture the features of large-dimensional graph-like structures such as ZX-diagrams. We also target the optimization of Clifford+T circuits, which allow for universal computation.

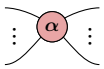
The rest of the article is structured as follows: Section 2 covers the state of the art of circuit optimization with ZX-Calculus. Section 3 presents the specifics of our ZX-RL optimization method. Section 4 reports the results of applying our method to random non-Clifford circuits and evaluates them against other ZX-based and Gate-based algorithms. Section 5 reports the results of extending our approach to structured quantum circuits for relevant applications. Finally, Section 6 summarizes the conclusions and potential future work.

2 Quantum Circuit Optimization via ZX-Calculus: State of the Art

2.1 General Overview

A ZX-diagram is a graphical representation of a linear map between qubits by means of an undirected graph [15, 16]. The basic elements of a ZX-diagram are spiders (nodes) and wires (edges). Spiders can be of two types: Z and X, and they can be interpreted as tensors composed of Pauli-Z and Pauli-X eigenstates, respectively. Using the graphical notation for ZX-Calculus they are represented as


$$:= \underbrace{|0\cdots0\rangle}_{m}\underbrace{\langle 0\cdots0|}_{n} + e^{i\alpha}\underbrace{|1\cdots1\rangle}_{m}\underbrace{\langle 1\cdots1|}_{n},$$


$$:= \underbrace{|+\cdots+\rangle}_{m}\underbrace{\langle +\cdots+|}_{n} + e^{i\alpha}\underbrace{|-\cdots-\rangle}_{m}\underbrace{\langle -\cdots-|}_{n}, \quad (1)$$

where m and n are the number of inputs and outputs of the spider, respectively, and α is a phase between 0 and 2π. The wires in the diagram can also be of two types, typically referred to as Simple and Hadamard wires[1]. A ZX-diagram can be simplified by applying a set of rules that preserve the underlying tensor representation of the diagram (see a summary of the rules in Figure 1). ZX-diagrams have a more open structure than quantum circuits, and their transformation rules are applicable regardless of the dimension of the spiders (tensors) involved. Notably, these rules can and give rise to transformations that can not always be described by single or two-qubit identities. A particularly relevant representation is expressing a ZX-diagram in its *graph-like* form [17]. The graph-like form of a ZX-diagram is such that

1. All spiders are Z-spiders (green spiders).
2. All connections between spiders are Hadamard wires (blue wires).
3. There are no parallel Hadamard edges or self-loops. (See Lemma 3.2 of [17] for an example.)
4. Every boundary spider i.e., an input or output qubit of the circuit is connected to at least one spider, and every spider is connected to at most one boundary spider.

[1] A Hadamard wire is a simple wire with a Hadamard gate. It is represented as a dashed wire painted in blue.

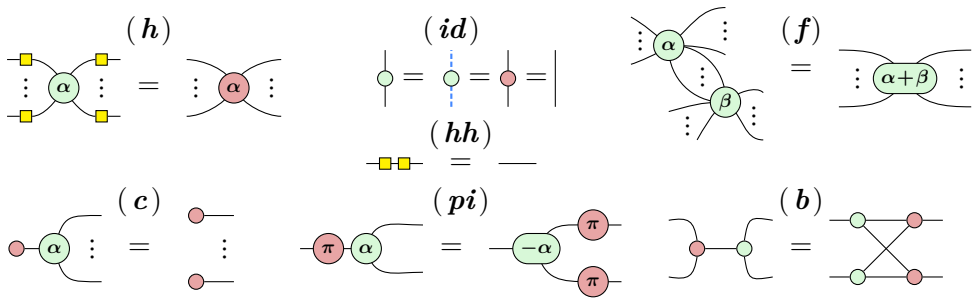


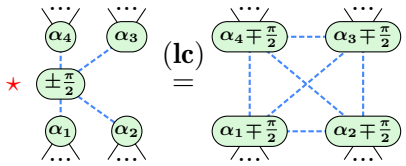
Figure 1: A summary of the ZX-Calculus. Note that '...' reads as any integer $n \in \mathbb{Z}$. The letters stand respectively for (***h***)adamard, (***id***)entity, (***hh***)-cancellation, spider-(***f***)usion, (***c***)opy, (***pi***)-commute and (***b***)ialgebra. These rules hold for green or red spiders, for any orientation of the diagram and up to non-zero scalars.

This transformation can always be achieved by iteratively applying the spider fusion and Hadamard rules (and additional rules derived in [17]). Once in its graph-like form, the diagram can be operated with a different set of rules, based on graph-theoretic simplifications, that are described in Section 2.2.

Quantum circuit optimization via ZX-Calculus involves the following steps: First, one should transform the quantum circuit into its equivalent ZX-diagram. This diagram is then converted into its *graph-like* form and simplified using graph-theoretic rules. After the simplification process is finished, one needs to transform the diagram back into an equivalent quantum circuit. This last step can be very inefficient or even unfeasible in some cases [18]. Even in the cases where this step is efficient, it can output circuits that are more computationally-expensive than the initial ones. This will be further discussed in Section 2.5. Although some heuristic rules have been suggested [19, 20], there is no known optimal strategy to tell which sequence of rule applications will yield the maximally optimized underlying circuit. The main focus of this paper is to improve the rule selection process using RL, rather than discovering new rules to simplify the diagram.

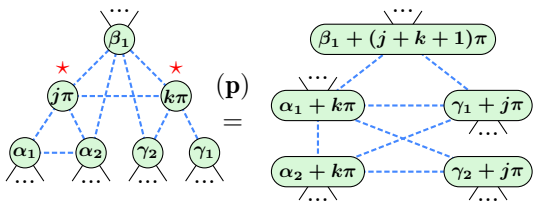
2.2 Graph-theoretic rules

Local complementation (Eq. 2) and pivoting (Eq. 3)[17] are the two essential rules that are used for the simplification of graph-like diagrams, and are inspired in their counterparts from graph theory [21, 22]. Local complementation (**lc**) can be applied to spiders whose phase is $\pm\pi/2$ (or proper Clifford spiders, marked with $\star$),



(2)

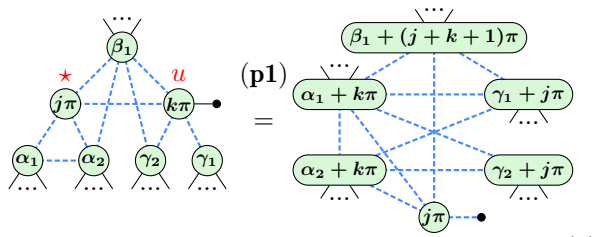
provided that they are connected by Hadamard wires to all of their neighbours and that they are all green spiders. This transformation removes the target spider from the diagram and modifies the connectivity of its neighbourhood by complementing it, i.e., two neighbours that were connected become disconnected, and two neighbours that were disconnected become connected. Additionally, the phases of the neighbours are updated by subtracting the phase of the removed spider. On the other hand, pivoting (**p**) can be applied to a pair of interior connected spiders with phase equal to 0 or π (Pauli spiders) that are only connected to green spiders through blue wires.



(3)

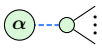
This rule transforms the graph by removing the pair of spiders at the cost of performing local complementation on three subsets: the unique neighbourhood of the first spider $\{\alpha_1, \alpha_2\}$, the unique neighbourhood of the second spider $\{\gamma_1, \gamma_2\}$ and the common neighbourhood of both spiders $\{\beta_1\}$. In short, any edge connecting nodes from different subsets will disappear, and vice versa. The phases of the remaining nodes are also updated as described in the aforementioned equation.

The pivoting rule can also be slightly modified to remove a spider (u) that is adjacent to a boundary spider (**p1**).



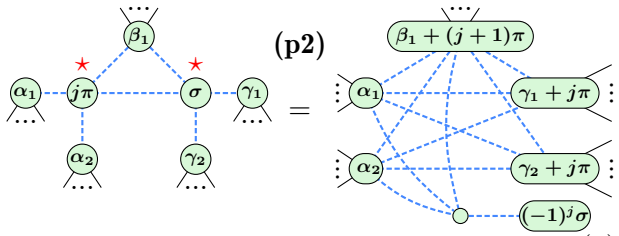
(4)

Further rules have been developed to simplify non-Clifford spiders (i.e., spiders with a phase that is not a multiple of $\frac{\pi}{2}$), allowing to address the optimization of universal quantum circuits via phase gadgets: an α-phase spider connected by a Hadamard edge to a phaseless spider.

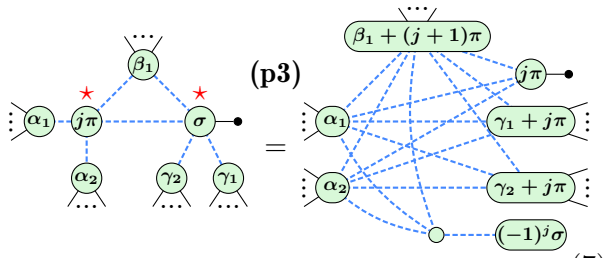


(5)

Phase gadgets enable the modification of both (**p**) and (**p1**) to work with non-Clifford spiders, at the expense of introducing a phase gadget after the application of each rule (Eq. (6) (**p2**), Eq. (7) (**p3**) respectively).

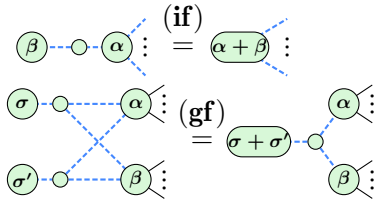


(6)



(7)

Additionally, phase gadgets can be simplified with two rules: (**if**) removes a phase gadget with a single leg and (**gf**) fuses two phase gadgets when they are connected to the same set of neighbours:



(8)

2.3 Circuit Extraction

Current circuit extraction algorithms make use of a necessary property for a deterministic graph state called *generalised flow* or *gflow* [23] from the Measurement Based Quantum Computing (MBQC) model [24]. Graph-like diagrams can be interpreted as an extension of a MBQC graph state where the phases of the spiders represent measurements on the XY, XZ or YZ of the Bloch sphere. In such diagrams, spiders are in the XY plane and phase gadgets are in the YZ plane. In [25] the authors developed a polynomial time extraction algorithm to extract a circuit from a graph state containing measurements in all three planes. Particularly, the algorithm extracts $\alpha-$spiders into a $R_Z(\alpha)$ gate and the Hadamard wires into either a Hadamard, Controlled-Z (CZ), or a combination of CNOT gates which is generally contingent on the connectivity of the graph. Furthermore, spiders in the XZ and the YZ plane need to be converted into spiders in the XY plane, which results in an addition of Hadamard wires during the extraction process. The existence of gflow in a transformed diagram is guaranteed if the initial diagram has gflow, as it is the case for quantum circuits, and the rules applied preserve it. Its exact calculation is not required for circuit extraction, the knowledge that one exists suffices. It should also be noted that the circuit extraction process can induce the addition of gates that can be trivially simplified afterwards with a gate-based optimizer. In this work, the extraction algorithm is treated as a black box that cannot be optimized, and we limit our action space to rules that satisfy the gflow condition, such as the ones described in Section 2.2.

2.4 Simplification algorithms

In this section, we review three simplification algorithms based on ZX-Calculus, each targeting the reduction of different types of spiders or gates in the resulting circuit. These algorithms will be used to benchmark our approach.

2.4.1 Simplification of interior Clifford spiders

In [17] the first algorithm for quantum circuit optimization using ZX-Calculus was introduced. Starting from the diagram in its graph-like form, the algorithm is applied as follows:

1. Apply (**lc**) to remove all interior spiders with phase $\pm\pi/2$.
2. Apply (**p, p1**) to remove adjacent pairs of spiders with phase 0 or π whether they are interior or one of them is not.
3. Apply (**p2, p3**) to remove adjacent pairs of Clifford and non-Clifford spiders (with both interior or one of them boundary).
4. Apply (**gf**) to further reduce non-Clifford spiders.

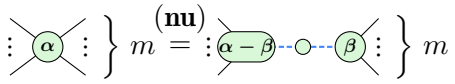
This sequence of rules is followed iteratively until there are no transformations available and the process terminates. In this regard, it is crucial to understand the trade-off between removing a spider and altering the connectivity of the diagram, as this may imply a reduction in the total number of gates but also increase the number of two-qubit gates (Figure 2b). This algorithm is implemented under the name of `full_reduce` in the library PyZX.

2.4.2 Simplification of non-Clifford spiders

In [26], the authors introduced an algorithm to optimize the T-count, i.e., the number of T gates required to implement the quantum circuit, using ZX-Calculus. The authors first parameterize the original circuit, $C[\alpha_1, ..., \alpha_n]$, where $\alpha_1, ...\alpha_n$ are variables representing the phases of the spiders. The phases are stored in a table $\tau : \{1, ..., n\} \to \mathbb{R}$ such that the original circuit can always be retrieved with $C[\tau]$. The algorithm then proceeds to run the `full_reduce` on $C[\tau]$ and *symbolically* tracks the phases after two variables are added together. Specifically, after the application of the rules (**if**) and (**gf**), depending on the sign difference between the affected variables, i, j, the table τ can be updated as: $\tau'(i) := \tau(i) \pm \tau(j), \tau'(j) := 0$ and $\tau'(k) := \tau(k)\ \forall k \notin \{i, j\}$, generating an equivalent ZX-diagram. After the termination of the `full_reduce`, the final circuit is obtained with $C[\tau']$, which will possibly contain fewer non-Clifford gates. This technique is denominated *phase teleportation*. By construction, the phase teleportation algorithm results in the same T-count as the `full_reduce`. However, phase teleportation does not change the original structure of the circuit, and in particular, the number or location of the two qubit gates. Therefore, employing phase teleportation as a initial processing in a compound simplification algorithm is rather convenient, as the further simplifications could potentially perform better since, in general, there will be less 'blockage' in the form of non-Clifford gates. This algorithm is implemented under the name of `teleport_reduce` in the library PyZX.

2.4.3 Simplification of two-qubit gates

As mentioned, it is extremely complicated to predict the exact number of two-qubit gates that will be obtained after the circuit extraction of ZX-diagrams preserving gflow. However, in [20] the authors try to do so using heuristics based on the number of Hadamard wires obtained after a rule application, precisely to avoid a highly connected ZX-diagram. To apply (**lc**) and (**p**) to spiders with arbitrary phase, the rules (**f, id**) are introduced, similar to what is done in (**p2**). They find that applying (**lc**) and (**p**) to general spiders introduces spiders in the XZ and YZ plane, which add a significant amount of Hadamard wires in the circuit extraction that can not be optimized with the heuristics used during the algorithm. To solve this issue, they also explore the use of the neighbour unfusion rule (**nu**) (with $|m| = 1$)



(9)

which turns out very effective in the reduction of two qubit gates but that does not always preserve gflow. The authors present a greedy algorithm, guided with heuristics, that selects (**lc**) and (**p**) until termination, which we denominated as `gflow-heur`. Summarizing, preserving gflow allows deviating from the rigid circuit structure and thus enhances exploration of graph-states where an effective optimization can be found. However, using this formulation, it is unclear how to predict the resulting amount of two-qubits gates without performing circuit extraction.

With this goal in mind, in [19] the authors introduce an optimization algorithm based on rules preserving *causal flow* or *cflow* which is a stricter condition than gflow, though not necessary, for a deterministic MBQC graph-state [27]. Even though only (**if**) is known to preserve cflow, there exists a computationally feasible algorithm introduced in [28] (which scales as $\mathcal{O}(|I||V|)$ compared to the gflow calculation algorithm that scales as

$\mathcal{O}(|V|^4)$, with I, V the Inputs and Vertex set) that allows the authors to compute cflow after a rule application, thus ensuring the cflow preservation throughout the simplification algorithm. The advantage of preserving cflow is precisely that there exists a direct equivalence between a graph-state and its underlying circuit, which results in a direct way of computing the two-qubit gate count, at the cost of deviating less from the rigid circuit structure and therefore leaving less room for exploration of more complex ZX-diagrams. The authors present a greedy algorithm that selects (**if, lc, p**) alone or combined with (**nu**) with $|m| = \{0, 1, 2, 3, 4, 5\}$ (to apply to general spiders) that prioritizes termination, which we denominate as `cflow-heur`.

2.5 General remarks and tests

To illustrate the performance of the aforementioned optimization algorithms, we conducted a test on random Clifford+T circuits of 10 qubits, with uniform probability of inclusion of each type of gate (S,T,CNOT,HAD). The first algorithm depicted is implemented under the name of `basic_optimization` in the PyZX library (`basic-opt` in the plot, illustrated in blue). It is a peephole optimizer that runs the circuit back-and-forth applying SWAP and gate cancellation identities until no further simplifications can be made. Note that since the extract circuit procedure usually adds gates combinations that can be trivially cancelled, this algorithm is a perfect fit as a post-processing step after the optimization. The second algorithm compared is the `full_reduce` (illustrated in purple) with the `basic_optimization` as a post-processing step. As mentioned, the application of all available rules until termination yields suboptimal results, significantly increasing the quantity of two-qubit gates for relatively shallow circuits (Figure 2b), and being outperformed by the simple gate-based optimizer (Figure 2a). The third and fourth algorithms are based on the `cflow-heur` and `gflow-heur` heuristics explained in Section 2.4.3 and are denoted as `cflow-zx` (in orange) and `gflow-zx` (in magenta) respectively. Both algorithms include the `basic_optimization` plus the `teleport_reduce` as a pre-processing step to simplify trivial identities and reduce the T-count. Additionally, the `basic_optimization` is included after circuit extraction as a post-processing step. The `cflow-zx` algorithm manages to outperform the previous algorithms by reducing both the number of single and two-qubit gates. Considering the existing results presented for the previous algorithms and the ones we obtained for the specific type of circuits under study, we choose both `gflow-zx` and `cflow-zx` algorithms to benchmark against our approach. The former allows for a fair comparison of the advantage that the RL approach can provide with the same action space and circuit extraction procedure, whilst the latter is chosen as, to the best of our knowledge, is the most competitive ZX-calculus based optimizer for two-qubit gate reduction.

3 RL-ZX Based Quantum Circuit Optimization

3.1 Proximal Policy Optimization

RL is a machine learning paradigm in which an agent learns, through trial and error, to perform a task on an environment. A training loop in RL follows a simple structure: The agent receives an observation s of the current state of the environment and picks its next action a upon a set of available ones using the experience gathered from previous attempts. Afterwards, the environment returns both a reward for the performed action and a new observation that can be used to update the agent strategy.

There are a myriad of RL algorithms that have found success in applications for many diverse fields. In this work, we use the *Proximal Policy Optimization* algorithm (PPO), which has a positive track record in similar circuit optimization settings. The PPO algorithm is a policy-gradient method which relies on the optimization of the parameters of a policy function $\pi(a|s)$. This function returns a probability distribution over all feasible actions. Additionally, the agent guides its learning process by interpolating a value function that estimates the expected value of the returns for a given state, $V(s)$. Both $\pi(a|s)$ and $V(s)$ functions are typically approximated using Deep Neural Networks (DNNs). The DNN used for approximating the policy function is often referred to as the *actor*, as it determines the optimal action in a given state. On the other hand,

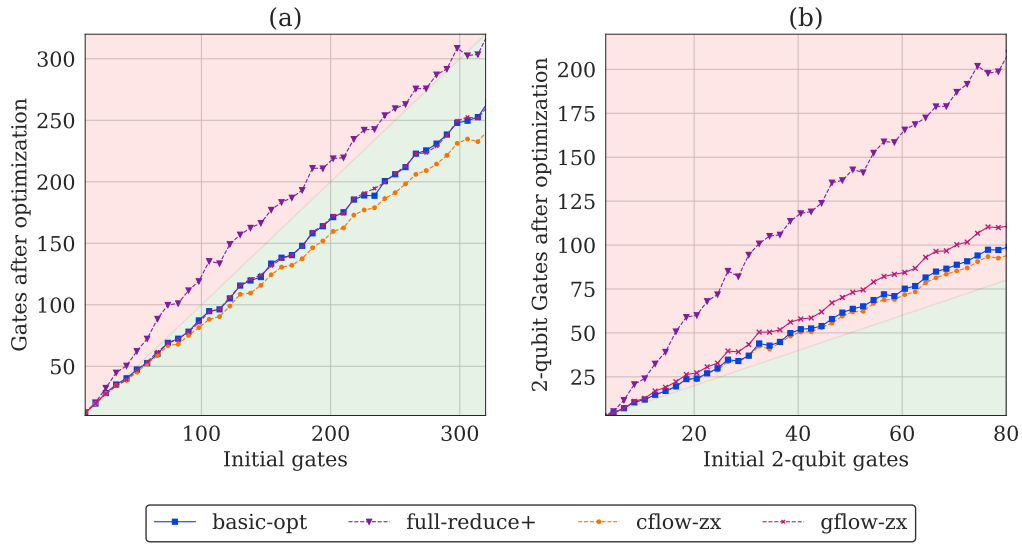


Figure 2: Total amount of gates after the optimization of quantum circuits using the `basic_optimization` algorithm (`basic-opt`, blue), the `full_reduce` algorithm (purple), the cflow preserving algorithm (`cflow-zx`, orange) and the gflow preserving algorithm (`gflow-zx`, magenta). Tests conducted on 10-qubit Clifford+T, with an increasing number of initial gates. Shaded regions in red indicate instances of unsuccessful compression, while those shaded in green denote successful compression. (a) Total Gate Count (b) Two-qubit Gate Count.

the DNN used for approximating the value function is known as the *critic*, as it evaluates the expected returns for a given state. This actor-critic architecture is a common approach in RL algorithms. During the training phase, the optimization of the parameters for both networks is done simultaneously by minimizing the loss function

$$L^{PPO}(\theta) = L^{Actor}(\theta) + c_1 L^{Critic}(\theta) - c_2 L^{Entropy}(\theta), \tag{10}$$

with $L^{Actor}(\theta)$, $L^{Critic}(\theta)$ and $L^{Entropy}(\theta)$ defined as

$$L^{Actor}(\theta) = \hat{E}_t \left[\max\left(-\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t, -\text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) A_t \right) \right], \tag{11}$$

$$L^{Critic}(\theta) = \frac{1}{2} \hat{E}_t \left[\max\left((V(s_t) - V_t^{\text{target}})^2, (V_{\text{clip}}(s_t) - V_t^{\text{target}})^2 \right) \right], \tag{12}$$

$$L^{Entropy}(\theta) = -\hat{E}_t \left[\sum_{i=1}^{n} \pi_\theta(a_i|s_t) \log(\pi_\theta(a_i|s_t)) \right], \tag{13}$$

The policy loss, $L^{Actor}(\theta)$ in Eq. 11, is computed as the product of the ratio of probability change between policies and the Generalized Advantage Estimator (A_t) [29] which, in simple words, measures how much better an action is compared to the average action at a given state. $\hat{E}_t$ indicates the empirical average over a finite batch of samples taken at different discrete time steps t within an episode. Hence, this term decreases if the new policy increases the probability of selecting actions with higher returns with respect to the previous one, and likewise, it reduces the probability of selecting non-beneficial actions. The critic loss, $L^{Critic}(\theta)$ (Eq. 12), essentially corresponds to the mean-squared error between the critic network's prediction and the actual return obtained by the agent (V_t^{target}). Hence, it is a measure of how well the critic can estimate the expected return given a state. Both terms can be slightly modified to include clipping restrictions (with V_{clip} referring to the clipped value function in Eq. 12) that limit the amount of change in each function at each step, so as to

achieve a more stable training.

The last term in the loss function, $L^{Entropy}(\theta)$ in Eq. 13, is included in order to balance exploration and exploitation during training. Exploration refers to the process of experimenting with novel actions that may potentially yield superior outcomes in subsequent stages. Conversely, exploitation involves adhering to the most advantageous known actions to optimize immediate rewards. An effective reinforcement learning agent should have sufficient exploration capacity in order to uncover new and improved actions, while also exploiting adequately to avoid wasting time and resources on the optimization of networks for the interpolation of suboptimal actions. In PPO, the entropy term is equivalent to the entropy of the log-probabilities produced by the actor network. Given that the entropy loss is subtracted in the total loss function, this term effectively guides the agent towards parameter configurations that increase the policy's uncertainty.

The PPO algorithm is known for its stability, as it avoids drastic policy updates that could lead to *catastrophic forgetting*, i.e., an abrupt change in the network that destroys the knowledge gained from previous experiences. However, perhaps its most significant advantage is its sample efficiency, as it allows for the parallelization of the sample generation process, which means that multiple instances can be run simultaneously. This feature significantly speeds up the learning process. For our particular task in hand, this proves to be very beneficial as it allows the agent to gain experience from many different circuit configurations, allowing for better generalization capabilities.

3.2 Graph Neural Networks

Graph Neural Networks (GNNs) are a type of artificial neural networks that are particularly well suited to work with data that can be represented as graphs, as it is the case for ZX-diagrams. These type of networks typically involve *message passing* layers that propagate the information of each node of the network to its nearest neighbours, i.e., the nodes that are connected to it. Hence, by iteratively adding several of these layers, the network is capable to capture long distance correlations in the data.
There are numerous variants of Graph Neural Networks (GNNs) in the existing literature, each employing a unique method to integrate the information received from a node's neighbours to update its current state. In this study, we utilize Graph Attention Networks (GATs), specifically GATv2 layers [30, 31]. These networks deviate from simpler GNNs by executing a weighted aggregation of information. The weights for this aggregation are computed using an attention layer in the calculation process. This allows the network to identify the significant neighbours of a node, namely those that contribute most significantly to the learning process. To elaborate further, let $\mathbf{x}_i^t$ denote the input features to the GATv2 message passing layer, and let $\mathcal{N}(i)$ represent the set of neighbouring nodes. The output features $\mathbf{x}_i^{t+1}$ are then computed as

$$\mathbf{x}_i^{t+1} = \alpha_{i,i}\mathbf{\Theta}_s\mathbf{x}_i^t + \sum_{j\in\mathcal{N}(i)} \alpha_{i,j}\mathbf{\Theta}_t\mathbf{x}_j^t, \tag{14}$$

where $\mathbf{\Theta}_s$ and $\mathbf{\Theta}_t$ are the learnable parameters to be optimized, and $\alpha_{i,j}$ represents the attention coefficient between node i and node j. In its turn, these attention coefficients are obtained as:

$$\alpha_{i,j} = \frac{\exp\left(\mathbf{a}^\top \mathcal{G}\left(\mathbf{\Theta}_s\mathbf{x}_i + \mathbf{\Theta}_t\mathbf{x}_j + \mathbf{\Theta}_e\mathbf{e}_{i,j}\right)\right)}{\sum_{k\in\mathcal{N}(i)\cup\{i\}} \exp\left(\mathbf{a}^\top \mathcal{G}\left(\mathbf{\Theta}_s\mathbf{x}_i + \mathbf{\Theta}_t\mathbf{x}_k + \mathbf{\Theta}_e\mathbf{e}_{i,k}\right)\right)}, \tag{15}$$

with $\mathbf{a}$, $\mathbf{\Theta}_s$, $\mathbf{\Theta}_t$ and $\mathbf{\Theta}_e$ additional weights that can be learned through the minimization of the target loss function. $\mathbf{e}_{i,j}$ represent the features of the edge connecting nodes i and j in the graph. Note that, unlike for the node features, these remain unchanged during the calculation. The $\mathcal{G}$ function corresponds to the LeakyReLU activation function. Finally, note that a Softmax normalization is used so that the sum of attention coefficients for each node always adds up to 1.

3.3 RL Agent: Structure and Learning Process

The schematic representation of the training process is depicted in Figure 3. Each training episode involves the generation of a random circuit, which is then transformed into a graph-like diagram. At each step, the agent selects an action from the available choices, based on the diagram's state. After the action is applied, the circuit is extracted from the modified diagram, and rewards are allocated based on the total number of gates obtained. The training utilizes the PPO algorithm, with both $\pi(a|s)$ and $V(s)$ networks

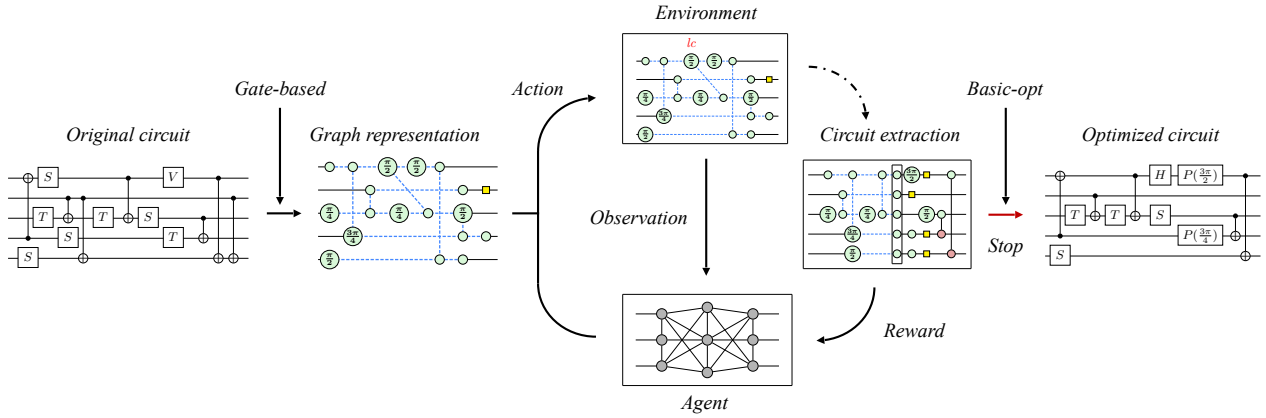


Figure 3: Overview of the `rl-zx` approach. From a random quantum circuit, the equivalent ZX-diagram in graph-like form is obtained. The agent uses its learned policy to choose between several graph transformations to be applied to the ZX-diagram (see Figure 4 for the details on the actor and the critic network structures). After the action is applied, the environment returns the resulting diagram to the agent as an observation, as well as its corresponding reward that is obtained after extracting the new circuit. The process is repeated until the agent decides to terminate the episode or no actions are available.

incorporating GATv2 layers. We apply both the `basic_optimization` and `teleport_reduce` algorithms before the agent starts acting on the circuit, which we laxly denominate as `gate-based`. The former removes trivial gate identities, reducing the workload of the agent. The latter reduces the amount of T gates in the circuit, which are hard to simplify with the action set at the agent's disposal.

The action space is restricted to local complementation (**lc**), pivoting (**p, p1, p2, p3**), gadget fusion (**gf**) and identity (**id**) rules only, with an additional action that the agent can select to terminate the episode (*STOP*). A detailed analysis of the architecture and methodology is required to effectively incorporate neighbour unfusion, as it is not guaranteed to preserve the *gflow*.

3.3.1 Actor and Critic

The actor layer's architecture comprises multiple *GATv2* layers. A layer configuration replicates the structure of the ZX-diagram, as depicted in Figure 4. Each green spider in the diagram is associated with a corresponding node in the network. In addition, every feasible rule that can be applied, defined by its type and the nodes that explicitly characterize it, is incorporated as a node within the network. These action nodes are connected solely to the nodes that define them and to the *STOP* action node. The connectivity pattern of the nodes derived from the green spiders is identical to that in the ZX-diagram. The *STOP* action is only linked to the other action nodes in the graph. Upon completion of the message passing layers, the feature vector of each action node, denoted as x_a^t, undergoes a transformation via the Softmax activation function. This process computes the probability of selecting a particular action, represented by $\pi(a_i|s)$, which is defined as

$$\pi(a_i|s) = \frac{\exp\left(x_{a_i}^t\right)}{\sum_{a\in\mathcal{F}(s)} \exp(x_a^t)}, \tag{16}$$

Here, $\mathcal{F}(s)$ denotes the set of all feasible actions applicable to the current state s of the ZX-diagram. Initially, each node j in the policy network is described by a 16-dimensional feature vector x_j^{init} with the following attributes: eight binary flags indicating the phase of the node if any (0, $\pi/4$, $\pi/2$, $3\pi/4$, π, $5\pi/4$, $3\pi/2$ or $7\pi/4$), three binary flags indicating whether the node represents an input, an output or a phase gadget, and 5 binary flags that identify whether the spider functions as a local complementation node, a pivoting node, a STOP node, an identity node or a gadget fusion node. These features collectively provide a comprehensive representation of the node's state within the network. For the critic network, the dimension of the feature vector is reduced to 11, as no action nodes are required. Additionally, each edge in the actor network is assigned a 6-dimensional *one-hot* vector that serves to distinguish the type of connection it represents, i.e., a wire in the ZX-diagram or a

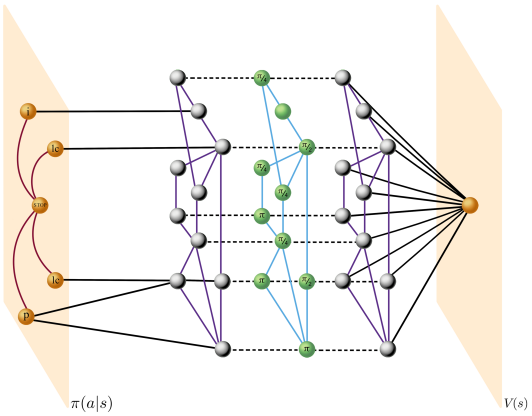


Figure 4: Schematic Overview of the Actor and Policy Networks: The policy network, denoted by $\pi(a|s)$, is visualized with each green spider from the diagram corresponding to a silver node, while the blue wires are depicted as purple connections. Action nodes are highlighted in orange, with connections to silver nodes and node STOP, which provide a complete description of the actions. The critic network shares a similar architecture; however, the orange node, representing the $V(s)$ output, is not part of the message passing layers. Instead, it is obtained after applying the global attention layer to the outputs of such layers. The self loops are not included for clarity.

connection between a node in the diagram and an action node of each given type. The term one-hot refers to a vector where only one element is "hot" (set to 1) while all other elements are "cold" (set to 0). Additionally, we also include self loops to each node as per the original formulation of the GATv2 layers [31].

3.3.2 Reward Function

After each action, the agent receives a reward that is equal to the difference between the amount of gates before the action is applied to the graph-like diagram and after. This reward is normalized by an expected compression factor that depends on the number of qubits and the initial number of gates of the circuit (30% of the expected amount of initial 2-qubit gates, in particular). We train the agent on two differentiated rewards: the first targets the total number of gates, whilst the second only considers the number of two-qubit gates. As explained above, the circuit extraction procedure can be quite inefficient, not only substantially increasing the amount of two-qubit gates in the circuit but also including single-qubit gates that can trivially be simplified afterwards. This overhead leads the agent to discard actions that achieve non-trivial simplifications. For this reason, the `basic_optimization` algorithm is applied before calculating the reward given to the agent. This inclusion slows down the training process, but for testing purposes, circuit extraction can be used only at the end of the episode, if desired. Additionally, we include a single shot reward at the end of the episode proportional to the difference in gates achieved by the agent and the `cflow-zx` optimizer for the same circuit. We find that, without this reward, controlling the balance between exploration and exploitation with only the entropy parameter is extremely hard, and the agent tends to rapidly converge to a policy of not applying actions to the circuit.

4 Experiments on Random Circuits

Establishing an appropriate and representative benchmark for evaluating our `rl-zx` strategy presents a significant challenge. The complexity arises from the fact that the results obtained not only depend on the agent's performance, but also on the rest of algorithms used in our workflow. These elements are treated as a black-box by the agent (i.e., the agent has no control nor information on its behaviour). This could lead to the conclusion that the agent's performance is suboptimal, even in scenarios where it had no opportunity to have a meaningful impact on the process. To partially circumvent this issue, we keep the best circuits obtained by the agent at any point during the optimization, not just at the end of the episode. This has the drawback of requiring the extraction of the circuit after each step in the optimization, penalizing the computational performance of our approach as a trade-off. In some cases, for which the agent is incapable of making any useful actions, the output circuit is directly the one obtained after the `basic_optimization` and `teleport_reduce` algorithms are applied. However, this is a very unlikely occurrence, as we will see below.

In evaluating our methodology, three criteria are considered: the quality of the outcomes, computational efficiency and the capacity of the agent to generalize to larger instances. These criteria are assessed against the previously selected benchmark. Notably, the `cflow-zx`

or preserving algorithm uses a different action space and a different circuit extraction method, but it is the best existing ZX-based optimization algorithm. Regarding scalability, it is noteworthy that while a trained RL agent can swiftly optimize circuits, the training phase itself can be time-intensive. Consequently, it is crucial to determine whether agents trained on smaller circuits, which require shorter training periods, are capable of effectively generalizing to larger-scale instances.

All trainings are done for randomly generated Clifford+T circuits and considering the two targets of optimization described above, i.e., the total amount of gates or two-qubit gates only. For the former, we train the agent on circuits of 5 qubits and 60 gates with equal probability of inclusion for each gate type. On average, these circuits have depth d $\approx$ 20. To study the capacity for the agent to generalize to larger circuits, we will both increase the number of qubits and the average depth of the randomized circuits. For the latter, the agent trains on circuits of 5 qubits and 70 gates, with increased probability of including CNOT gates to $\frac{1}{3}$. Both the increase in the amount of gates and two-qubit probability of inclusion are meant to compensate for the fact that when changing the target of optimization to two-qubit gates, the amount of meaningful actions (those that result in an improvement on the target) is reduced.

Trainings are done in a single node with 8 CPUs and a single GPU and take $\sim$ 16 hours. The agent and the environment are implemented using PyTorch [32] and Gym [33], respectively. For the agent, both the actor and critic contain 5 *GATv2* message-passing layers, with input/output channel dimension of 32. The critic network incorporates a global attention layer [34] following the *GATv2* layers. The additional hyperparameters used during the training are detailed in Table 1.

Table 1: PPO hyperparameters used to train our agent

Parameter	Value
Num. steps	512
Num. environments	8
Learning rate (η)	2×10^{-4}
Num. epochs	8
Minibatch size	512
Discount (γ)	0.99
GAE parameter (λ)	0.95
VF coeff. c_1 (Eq. 10)	0.5
Entropy coeff. c_2 (Eq. 10)	0.01
Clipping parameter. ϵ	0.1

4.1 Analysis of the Optimization Policy

We first focus on understanding the differences between the learnt optimization strategies for both experiments. To do so, we test the obtained agent policies on 1000 random circuits of increasing number of qubits (5, 10, 20, 40 and 80) and increasing average depth (d, 2d, 3d, 4d). For clarity, the largest instances are generated for 80 qubit circuits of 2100 random gates (which result on an average depth of 80). The probabilities of inclusion of each gate remain unchanged with respect to the training phase.

When optimizing the total amount of gates in the circuit, the agent consistently increases the amount of simplified gates across both circuit depth and number of qubits (Figure 5a). Naively, this signals that the agent is able to generalize correctly. On the other hand, the agent's performance seems to be mostly dependent on circuit depth when trying to reduce the amount of two-qubit gates (Figure 5b). However, when comparing the amount of actions that the agent performs before reaching the optimal circuit in an episode, this number increases across both circuit depth and number of qubits for both policies, until reaching the maximum length allowed, which varies between 50 and 100 depending on circuit size (Figure 6). To understand the scaling disparity between the optimal episode length and the compression achieved for each objective, we analyse the reward obtained per rule type for both optimization objectives, and find that local complementation is not impactful when optimizing two-qubit gates. This is seen by plotting the histogram of the change in gates produced by each action for all episodes (which we do not include for brevity). Hence, when targeting two-qubit gate reduction, the agent relies almost exclusively on pivotings, whilst local complementation can be interpreted as equivalent to a *PASS* action, that neither benefits nor penalizes

the agent immediately, although it produces a change in connectivity that may result in additional pivotings appearing next.

We plot the average number of actions of each type that are selected by the agent for both policies (Figure 7). We observe that whilst local complementation rules do appear to follow the same monotonic increase with both circuit size dimensions, the agent struggles to find beneficial pivotings to apply to the circuit as the number of qubits grows (Figure 7b), resulting in the performance disparity for both optimization objectives. This can be understood by the fact that pivotings affect the neighbourhood of two-spiders, leading to drastic changes of connectivity in the diagram after applied. This variance is dependent on the number of wires a spider can have, which is directly linked to the dimension of the Hilbert space (i.e. the number of qubits) when preserving gflow. Hence, modelling the effect of pivotings on the circuit extraction process becomes particularly difficult, specially considering that the former is treated as a black-box. One could try to include new features into the observation given to the agent to partially circumvent this issue, such as the heuristics that are used in [20]. We leave this as future work. Nonetheless, our findings are in accordance with the general conception that managing two-qubit gates reduction is the major limiting factor for ZX optimizers based on the preservation of gflow due to the inefficient circuit extraction process.

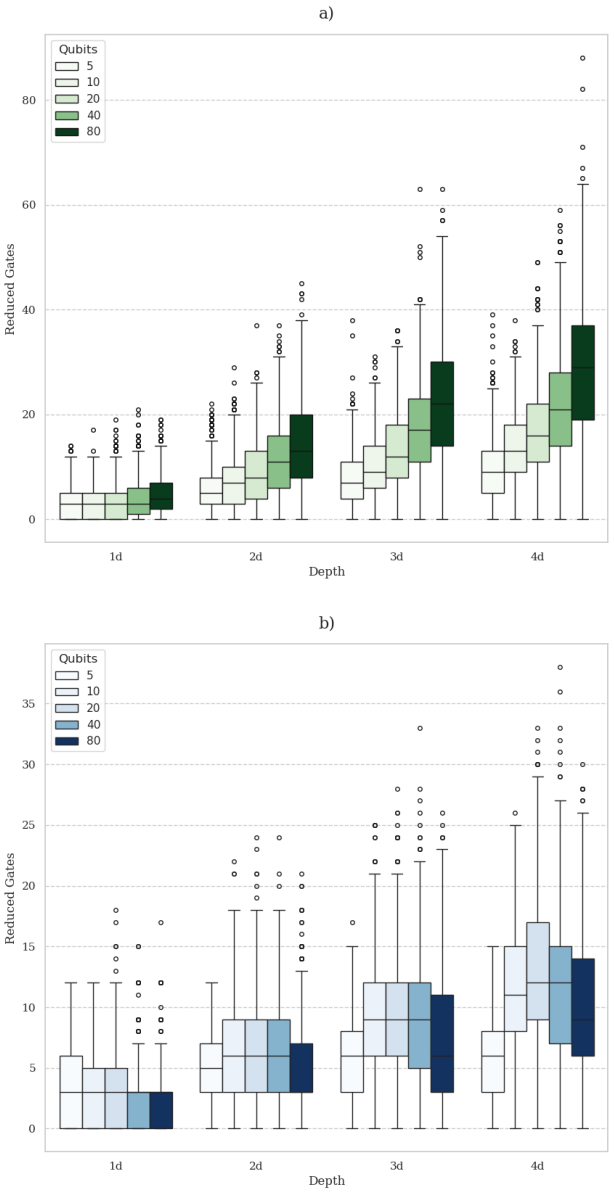


Figure 5: Box plot of the number of gates reduced by the `rl-zx` agent once trained. The initial value for comparison corresponds to the amount of gates after `basic_optimization` and `teleport_reduce` algorithms are applied to the circuit. Tests are performed for two-different tasks and across several circuit sizes, both in terms of number of qubits and average circuit depth. Statistics are drawn from one-thousand executions of random circuits for each circuit size and task. (a) Reduced single-qubit and two-qubit gates. Different shades of green depict different amount of qubits. (b) Number of two-qubit gates reduced. Different shades of blue depict different amount of qubits.

4.2 Quality of the compression

We assess the quality of the results obtained by our approach by comparing it against the `gflow-zx` algorithm. As previously mentioned, we consider the best circuit seen during an episode as the output of the `rl-zx` agent. This is relevant when increasing the size of the tests, as any misstep by the agent rapidly increases the amount of gates in the circuit.

Firstly, we study the frequency with which each optimizer achieves the best result, focusing on two-qubit count reduction due to being the most relevant task. Again, tests involve one thousand episodes per circuit size. The RL based approach consistently outperforms the competition for circuits of up to sixteen times in qubits and four times in average depth with respect to the

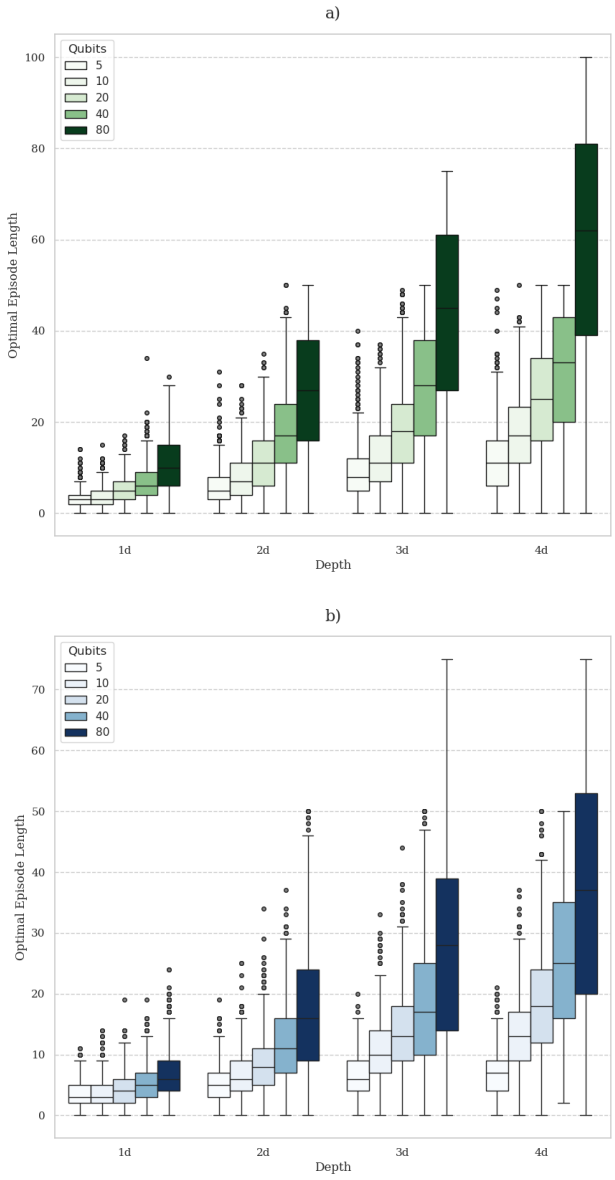


Figure 6: Box plot of the number of actions taken during an episode by the `rl-zx` once trained. Results are drawn from the same tests used to depict the amount of gates reduced. Different shades of green and blue depict different amount of qubits. (a) Actions performed when trying to reduce both single-qubit and two-qubit gates. (b) Actions performed when trying to reduce two-qubit gates.

amount of training (5 qubits and depth 20) (Figure 8). In particular, for the largest circuits, the obtained circuit is improved in above 80% of the cases. This demonstrates the utility of the approach in the regime of circuits that are expected to allow for quantum advantage experiments.
Somewhat counterintuitively, the agent performs poorly with respect to the gflow heuristic for 5-qubit circuits of large depth. This may be a consequence of the fact that the agent learns a "risk-free" policy, due to the high variance of the effect of an action, that is able to generalize well. Another factor to take into account is the fact that with the current observation features and the message-function implemented by *GATv2* layers, the agent has no information in regard to "extensive" properties of the graph (for instance, total number of nodes or total number of edges). We have also briefly explored using graph convolutional layers and aggregating node features for the critic network using the sum function instead of a global attention layer to capture this extensivity. However, our preliminary tests indicate that, even though results for smaller scale circuits improve, this architecture generalizes worse than our selected one. We leave the exploration on efficient ways of incorporating these features into the network as a future research line. As per the magnitude of improvement, we plot the distribution of the difference in gates between both approaches (Figure 9).On average,the improvement ranges between a 5x and 12x two-qubit gate reduction factor, increasing with circuit size. All the aforementioned results demonstrate the ability of the agent `rl-zx` to generalize its learned strategies to circuits of much larger size, which is crucial to ensure the scalability of the approach. However, this generalization is so far only demonstrated on circuits that follow the same random generation procedure and distribution, and most notably, the results obtained on such circuits are consistently worse (in almost 100% of the evaluated random circuits) than the ones obtained by the `cflow-zx` algorithm.

4.3 Computational Cost

Even though we have seen that our approach is able to improve the two-qubit gate compression with respect to state of the art gflow-based optimizers, it is important to understand the price to be paid in terms of computational cost to achieve

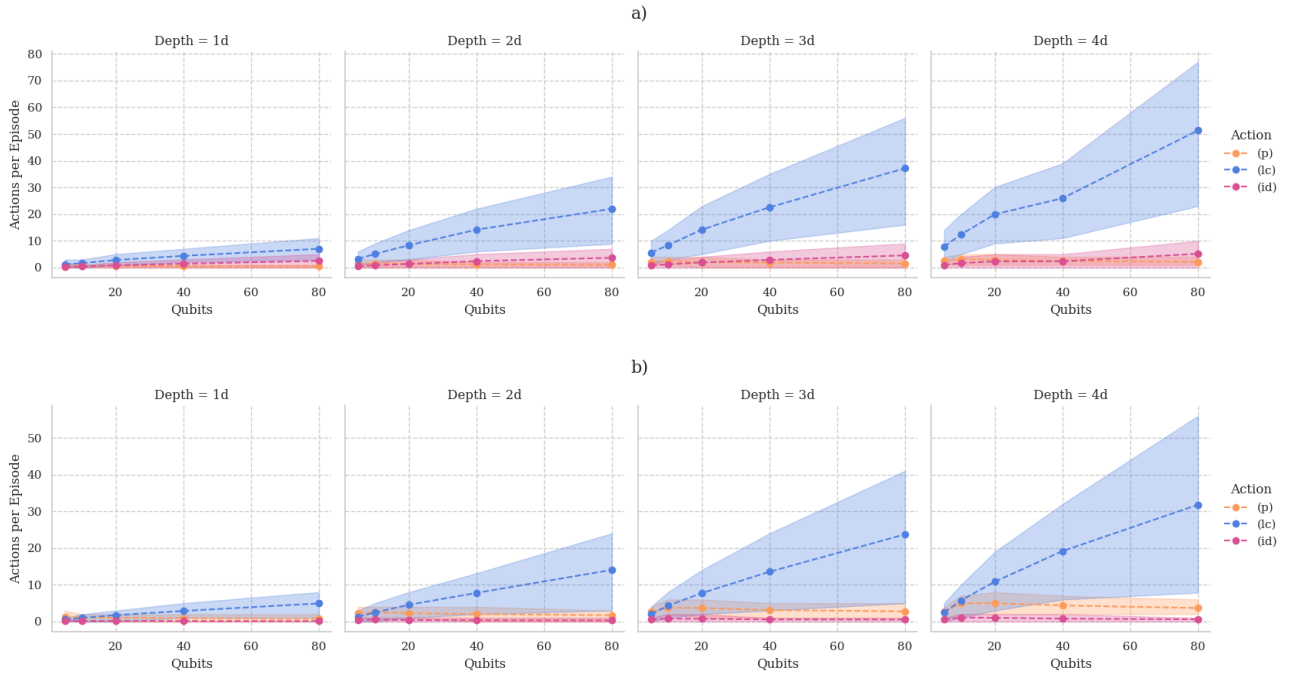


Figure 7: Average number of actions of each type performed per episode by the agent. For clarity, only pivotings (***p***) (orange), local complementations (***lc***) (blue), and identity rules (***id***) (purple) are considered. The non-represented actions are scarcely selected by the agent. Shaded regions represent the interval between percentile 15 and 85 of the distribution. Results are drawn from the same instances. (a) Policy learnt for single-qubit and two-qubit gate reduction. (b) Policy learnt for only two-qubit gate reduction.

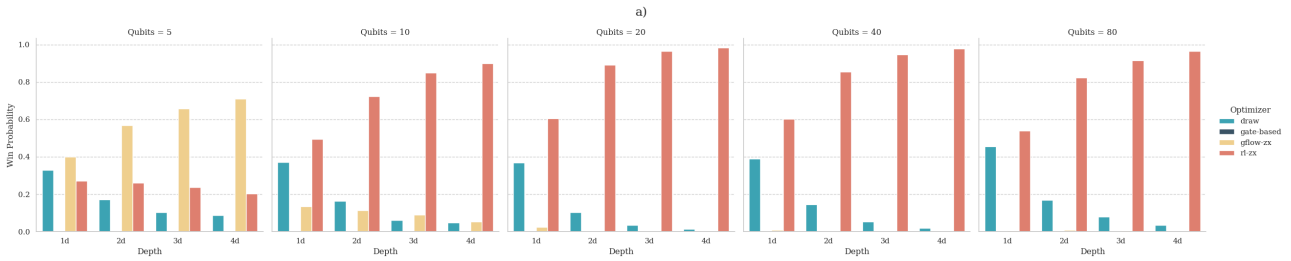


Figure 8: Percentage of test instances for which each optimizer achieves the best two-qubit gate compression, denoted as "Win Probability" for brevity. Results are shown for the same 1000 instances per circuit size and task. Our approach, `rl-zx`, is depicted in orange. 'gate-based' (dark blue) refers to the results after `basic_optimization` and `teleport_reduce` are applied to the initial random circuit, and `gflow-zx` is depicted in yellow. Circuits for which `rl-zx` ties with either of its competitors are represented as "draw" (light blue).

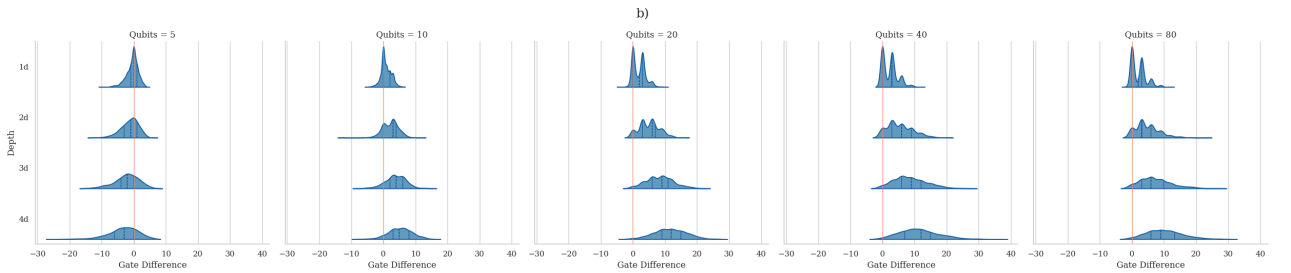


Figure 9: Distribution of the difference in 2-qubit gates between `gflow-zx` and our approach for both optimizations tasks and all tested circuit sizes. A positive gate difference implies that `rl-zx` achieves a better compression. Vertical lines, in red, signal a tie between both algorithms. Coloured dashed lines mark the quartiles of the distribution.

that. In the NISQ era, reducing the amount of gates is not only important to speed up the calculation, but particularly to reduce the probability of errors occurring during the execution. In this sense, the trade-off between the increased execution time of our algorithm and the gain in "quantum computing time" is not one-to-one. Nonetheless, we crudely plot the difference in execution time between the `gflow-zx` algorithm and our approach in Figure 10. We consider the full execution time of the algorithm, until the agent selects the STOP action or there are no actions remaining, and not the time required to reach the optimal circuit seen during an episode. We observe that our agent is consistently 4x-10x slower than the gflow heuristic, though with a subexponential scaling (at least for the studied circuits sizes) with both circuit depth and number of qubits.

Aside from further research on the agent's capacity to stop at the optimal time, there are several improvements in terms of the implementation of our solution that can be done to improve on these results. For instance, we have performed naive adaptations of the PyZX package to implement the environment, but one could carefully design the action matching procedure (i.e. identifying the actions that are feasible in a diagram) such that it is not applied to the whole graph after each step in an episode, but only on the neighbourhood of the spiders that have been transformed. In addition, much more sophisticated HPC approaches could be explored to parallelize execution. In terms of memory, the training on 5 qubit circuits of 60-70 gates requires 2GB of VRAM, for the specific batch size used, and execution of our largest circuits requires less than 1 GB.

5 Experiments on Structured Quantum Circuits

In the previous section, we have seen that the agent is able to generalize the learned strategies to much larger circuits. However, the selected workflow of optimization is not able to improve on the results of the `cflow-zx` algorithm, which is likely a result of the inefficient circuit extraction process. Considering the utility of our approach, we next modify the optimization workflow to include the `cflow-zx` algorithm at the end of the gflow-based optimization, including an additional reward to the agent that is equal to the difference in gates between applying `cflow-zx` to the final circuit after the agent selects the action STOP and simply applying `cflow-zx` to the initial circuit. This signals the versatility of the RL algorithm; the gflow-based action selection can naturally be integrated in hybrid optimization algorithms, which use other ZX-based rule sets and circuit extraction procedures, or even additional gate-based optimizers.

With the selected reward, we are biasing the agent toward policies that prioritize reducing the number of two-qubit gates using the gflow-preserving action set, while simultaneously enabling improved circuit performance when the `cflow-zx` is applied. The motivation behind this heuristic stems from previous findings [20] indicating that managing two-qubit gate count with gflow-preserving action-sets alone is highly complex, since the generated diagrams represent unitaries through spiders (tensors) of much larger dimensions than single and two-qubit gates. This allows for elaborate simplifications to be found, but also results in inefficient circuit conversions. These are precisely the gates that we aim to simplify with the rewarded action-selection process, while ensuring that they do not impact negatively to latter optimization routines (in this case only `cflow-zx`, but more complex workflows could also be considered).

As we will see below, the bias in the resulting policy can be detrimental for certain circuits. Specifically, in some cases, significantly increasing the number of two-qubit gates after applying gflow-preserving actions could lead to circuit configurations more amenable to subsequent cflow-based optimization.

To explore alternative approaches, we tested workflows that eliminate this bias, such as determining the reward only by the performance of the `cflow-zx` algorithm after every action, rather than only at the end of the episode. However, this strategy proved highly detrimental to the agent's generalization capabilities. We hypothesize that the reason for this outcome is that the agent receives no direct observation on the specific diagram to be optimized by the cflow algorithm. Since gflow-preserving actions do not inherently preserve cflow, applying`cflow-zx`

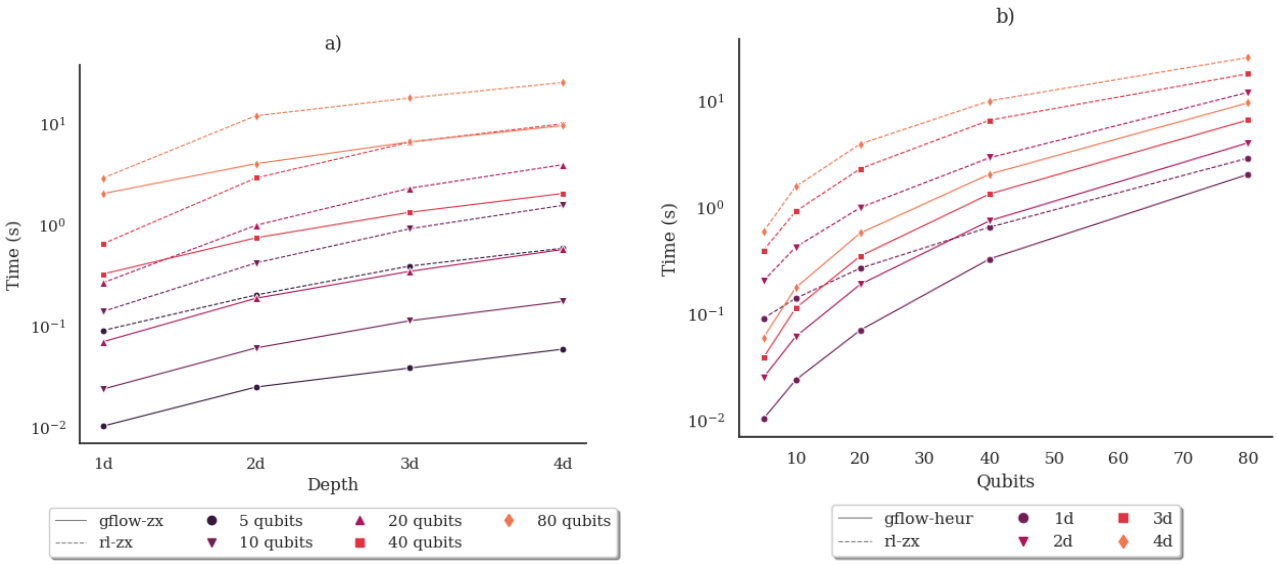


Figure 10: Comparison of the average episode execution time, in seconds, for the `gflow-zx` and our approach (dashed lines), with color code dependent on each relevant circuit dimension. (a) Scaling with respect to the average circuit depth. (b) Scaling with respect to the amount of qubits.

after each action requires extracting the circuit, converting the resulting circuit back to a graph-like form, and then optimizing it again, a computationally expensive process that interrupts the training flow. Additionally, the most challenging task for the agent is determining when to select the STOP action. This difficulty arises because the optimization policy is probabilistic, and as the circuit size increases, so does the number of possible actions, which rapidly reduces the likelihood of correctly selecting the STOP action, with the added drawback that a single "mistake" in selecting a pivoting instead of stopping with high dimensionality diagrams can drastically change the amount of two-qubit gates. We empirically find that the agent performs many more suboptimal episode steps and much more frequently with a purely "cflow-based" reward, likely also a consequence of the lack of information. The selected reward and workflow address these challenges, sacrificing part of the optimization potential through biases that allow for a more computationally efficient approach. In addition, this allows for the added benefit that with this workflow and at inference time, one could track and retain the best circuit observed throughout the gflow-preserving optimization process and then apply the `cflow-zx` algorithm to it, rather than relying solely on the final circuit produced after the STOP action. We recommend this approach if the user is particularly concerned by the stability of the optimization, as it is the case in production environments.

With this new workflow, we retrained the agent for random circuits of 5 qubits and 70 gates, using the same random circuit generator detailed in the previous section. After training, the resulting policy is tested on structured quantum circuits for relevant applications, e.g., the quantum Fourier transform [35]. The dataset used is standard in the literature and includes circuits of different sizes, and gate distributions. For these circuits, we aim to achieve the largest compression possible, and hence allow the agent multiple attempts, i.e. 1000 tries, at optimizing the same circuit, using its learned probabilistic policy. We benchmark the results against a gate-based state-of-the-art optimizer, the `NRSCM` [36], the `cflow-zx`, and a combination of the `gflow-zx` plus the `cflow-zx`. Furthermore, and since the aforementioned algorithms are deterministic, we also assess the relevance of the results against a probabilistic approach (`random-zx`) that selects actions randomly during the gflow-preserving optimization step, including the `cflow-zx` at the end of the optimization as well. The results are shown in Table 2 and discussed below.
Notably, a combination of gflow + cflow opti-

mizers can achieve optimal results in almost all cases, with or without any intelligent action selection. This signals that a refinement in the rule selection process for these optimizers can indeed circumvent the limitations identified for the gflow-based optimizers alone. However, profiting from this combination is by no means trivial, as demonstrated by the fact that a combination of the most advanced gflow-based heuristic with the `cflow-zx` does not improve the results in the majority of cases. The `rl-zx` algorithm proves to be a consistent enhancement to the `cflow-zx` algorithm, but there is still margin of improvement, as a fully random action selection is eventually able to find even optimal circuits for several instances. In its turn, the probability of finding these circuits randomly is very small and decreases as the size of the circuit increases (see Figure 11). Not only that, for some of the largest circuits, the random algorithm did not terminate within the maximum job time allowed (72h). In addition, we expect the RL-based approach to improve its performance if more similar or larger circuits are used for training. This can be very useful for experiments that are willing to spend classical resources to squeeze the maximum performance out of the quantum algorithm, for instance, when attempting to push the state-of-the-art precision of some relevant quantum calculation. Several successful use cases along these lines have been reported in the optimization of variational ansatzes for quantum chemistry applications [37–39]. Finally, we also analyse the ability of the `rl-zx` algorithm to be used as a single-shot deterministic optimization tool. To do so, the policy is modified such that the action selected corresponds to the one with the largest probability. With these approaches, the agent achieves more modest improvements in the results of `cflow-zx` and in a smaller number of instances (13 of the 33 circuits tested), see Table 3.

6 Conclusions & Future work

This work presents a new RL approach to quantum circuit optimization that takes advantage of ZX-Calculus by using a more sophisticated scheme based on Graph NNs instead of convolutional layers. To assess the validity of the method, we present results across three relevant criteria: quality of the optimization, computational efficiency and scalability. These results are benchmarked for Clifford+T circuits against the best-performing gflow based circuit optimization algorithm across two differentiated optimization objectives, the total amount of gates in the circuit and two-qubit gates alone.

We demonstrate that the agent is able to generalize the learned strategies and outperform the competition for circuits of up to 80 qubits and 2100 gates for both tasks. However, the obtained compression for these circuits is consistently worse than the one achieved by the `cflow-zx`. This difference in performance is to be expected, as the gflow-based extraction process is known to be very inefficient when dealing with two-qubit gates. In contrast, the cflow formulation allows estimating the amount of gates resulting from the extraction of each spider in the diagram, simplifying the decision process. Not only that, the `cflow-zx` algorithm allows for diagram transformations that increase the amount of spiders in the diagram, through unfusion rules, achieving a larger space of candidate diagram configurations.

To improve on the previous results, we modify the optimization workflow to include the `cflow-zx` algorithm at the end of the optimization, after the gflow-preserving transformations are applied by the agent. The reward given to the agent during training is also modified to target actions that achieve simplifications through the gflow-based formulation that are unattainable by the cflow-based one, without hindering the latter's performance. This is a clear example of the versatility of the approach; the agent is able to adapt its policy to enhance the optimization result in a workflow that combines a simple gate-based optimization algorithm as well as a more complex ZX-based heuristic. This versatility is particularly useful for current experimental platforms, as the reward function can be shaped taking into account the specific properties of the quantum hardware in which the circuit will be executed, e.g., by weighting each type of gate depending on its fidelity, or taking into account qubit coherence times and the depth of the circuit. An unrefined implementation of the agent is only an order of magnitude slower than the `cflow-zx` algorithm optimizing circuits, but with a similar subexponential scaling. A full training

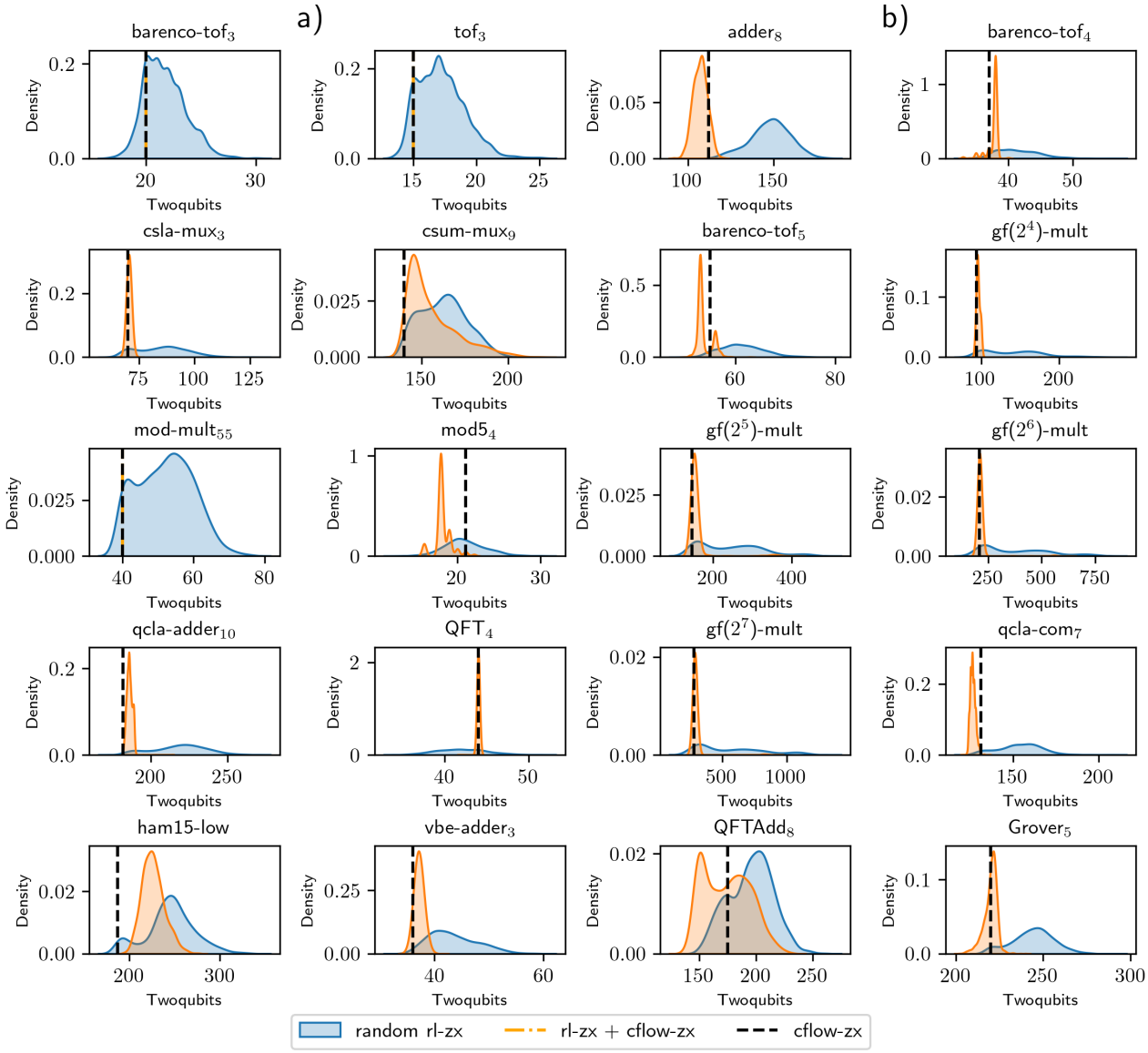


Figure 11: Comparison of the two qubit gates distribution obtained for the for `rl-zx + cflow-zx` (orange) and the `random-zx` (in blue) for different circuits. In a dashed black line is depicted the two qubit gates obtained by the `cflow-zx`. a) First two columns. Circuits for which an execution of the random policy achieved the optimal result. (b) Last two columns. Circuits for which an execution of the `rl-zx + cflow-zx` achieved the optimal result.

	Original			NRSCM[35]		rl-zx + cflow-zx		cflow-zx[19]		random-zx		gflow-zx[20] + cflow-zx	
Circuits	**Qubits**	**Total Gates**	**2Q**	**Single Qubit**	**2Q**	**Single Qubit**	**2Q**	**Single Qubit**	**2Q**	**Single Qubit**	**2Q**	**Single Qubit**	**2Q**
$adder_8$	24	900	409	315	291	261	**244**	312	267	-	-	305	278
Adder8	23	637	243	96	94	122	**92**	125	112	142	116	156	141
barenco-tof_3	5	58	24	22	18	25	20	25	20	25	**17**	25	20
barenco-tof_4	7	114	48	38	34	45	**33**	52	37	39	34	52	37
barenco-tof_5	9	170	72	54	**50**	67	**50**	66	55	65	**50**	67	60
barenco-tof_{10}	19	450	192	134	**130**	162	**130**	174	146	179	138	169	147
tof_3	5	39	18	21	**14**	21	15	21	15	21	**14**	21	15
tof_4	7	75	30	33	**22**	33	24	33	24	39	23	33	24
tof_5	9	105	42	45	**30**	45	33	45	33	50	31	45	33
tof_{10}	19	255	102	105	**70**	106	77	105	78	105	78	105	78
csla-mux_3	15	170	80	85	70	88	68	81	70	83	**65**	91	70
csum-mux_9	30	448	168	126	140	146	140	148	140	148	**138**	147	142
gf(2^4)-mult	12	243	99	88	99	80	**91**	80	94	81	93	80	94
gf(2^5)-mult	15	379	154	142	154	128	**145**	128	146	128	147	128	148
gf(2^6)-mult	18	545	221	182	221	179	**202**	179	209	179	208	179	209
gf(2^7)-mult	21	741	300	225	300	237	**278**	237	283	237	282	237	284
gf(2^8)-mult	24	981	405	307	405	288	**379**	288	383	-	-	288	390
mod-mult-55	9	119	48	51	40	48	40	48	40	48	**39**	48	40
mod-red-21	11	278	105	103	**77**	106	**77**	110	83	109	80	115	85
$mod5_4$	5	63	28	23	28	13	16	19	21	15	**13**	17	21
qcla-$adder_{10}$	36	521	233	216	183	219	182	220	**174**	219	179	226	185
qcla-com_7	24	443	186	152	132	158	**120**	156	131	158	125	159	129
qcla-mod_7	26	884	382	332	**292**	372	296	372	**292**	377	297	377	296
rc-$adder_6$	14	200	93	69	71	89	68	84	71	104	**67**	88	71
vbe-$adder_3$	10	150	70	39	50	50	35	50	36	48	**34**	50	36
$QFTAdd_8$*	16	476	184	138	184	135	**138**	126	175	140	149	129	157
QFT_4	5	187	46	-	-	113	42	115	44	112	**35**	115	42
QFT_8	8	148	56	-	-	56	**42**	-	**42**	58	**42**	58	**42**
QFT_{16}	16	586	228	-	-	183	**144**	-	**144**	171	155	184	**144**
QFT_{32}	32	1562	612	-	-	456	**368**	-	**368**	-	-	456	**368**
ham15-low*	17	443	236	-	-	167	195	157	187	155	**186**	158	203
hwb6*	7	259	116	-	-	110	**90**	111	98	117	91	113	**90**
$Grover_5$*	9	831	288	-	-	287	**207**	281	220	291	217	285	224

Table 2: Comparison between the results obtained by several optimizers on structure quantum circuits for relevant applications. Results in bold show the minimum amount of 2-qubit gates observed for each circuit. Additionally, circuits for which the `rl-zx` optimizer improves on `cflow-zx` alone are referenced with an asterisk. The equivalence between initial and final circuits has been validated for all circuits excepting QFT16, QFT32 and QFTAdd8, due to limitations in computational resources. However, cflow and gflow preservation should guarantee its validity.

Circuits	rl-zx-det	cflow-zx
$adder_8$	266	274
Adder8	109	112
barenco-tof_4	33	37
barenco-tof_5	53	55
barenco-tof_{10}	137	146
gf(2^4)-mult	91	91
gf(2^5)-mult	142	145
gf(2^6)-mult	203	202
gf(2^7)-mult	278	278
gf(2^8)-mult	379	383
qcla-com_7	125	133
QFTAdd8	157	175
hwb6	91	98
$Grover_5$	195	220

Table 3: Comparison between the two-qubit gates obtained by the deterministic rl-zx policy, `rl-zx-det`, and the `cflow-zx`, only including those circuits for which there is a gain.

process takes around 16 h on a small server.

With this improved workflow, the agent is able to improve the state-of-the art of structured quantum circuits that correspond to relevant applications, even in comparison to gate-based algorithms, with circuit dimensions much larger than the ones observed during training and including much different gate distributions. However, these results need to be partially contextualized: due to the probabilistic nature of the policy, the agent requires multiple attempts at optimizing the same circuit to achieve the absolute best results. This makes our approach particularly interesting for applications in which it is crucial to optimize quantum resources, even at the expense of classical ones, a reality in many of today's practical use cases. A viable alternative is to modify the selection policy by always picking the action with the largest probability, resulting in a deterministic behaviour oriented towards robust and continuous use. With this, the agent is still able to moderately improve the compression

of several of the previous circuits.
A possible extension to our work would be to allow the agent to select the actions applied at different stages of the optimization workflow. For instance, during the `teleport_reduce`, to target T-gate simplification, or the `cflow-zx` as well. Nevertheless, extending the action space to allow for unfusion rules will require ensuring the preservation of flow after each action is applied, increasing the computational cost of the solution. In this regard, if unfusion actions are included, one should prioritize utilizing the cflow based formulation, since ensuring flow preservation is less computationally intensive. Another likely advantage of the cflow formulation is that it would facilitate including features in the agent's observation that inform the agent on the effect of the circuit extraction process for a given diagram. This could both simplify the training phase and improve performance overall. Finally, it is worth noting that further research on the observation features and the overall agent architecture could also improve the agent's decision on when to stop the optimization, which would greatly benefit the computational performance of the algorithm, as less optimization steps would be required and the circuit extraction process could be used only at the end of the episode.
Upon completion of this work, we found reference [40] where the authors use a RL approach for spider-count reduction in ZX-diagrams and reference [41], where circuit optimization is done using Transformers.

7 Data and code availability

Data and code to verify and replicate our results can be found in [42]. Other findings from this study are available from the authors under due request.

Acknowledgments

We thank Professor R.Rey and the Theory&Applications team at Qilimanjaro for their constructive criticism and very helpful advice. A.G-S received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 951911 (AI4Media). This work was supported by the Agència de Gestió d'Ajuts Universitaris i de Recerca through the DI grant (No. 2020-DI00063) and by MICIU/AEI/10.13039/501100011033/ FEDER, UE.

References

[1] John Preskill. "Quantum computing in the NISQ era and beyond". Quantum **2**, 79 (2018).

[2] Beatrice Nash, Vlad Gheorghiu, and Michele Mosca. "Quantum circuit optimizations for nisq architectures". Quantum Science and Technology **5**, 025010 (2020).

[3] Scott Aaronson and Daniel Gottesman. "Improved simulation of stabilizer circuits". Phys. Rev. A **70**, 052328 (2004).

[4] Vadym Kliuchnikov and Dmitri Maslov. "Optimization of clifford circuits". Phys. Rev. A **88**, 052307 (2013).

[5] Richard S. Sutton and Andrew G. Barto. "Reinforcement learning: An introduction". The MIT Press. (2018). Second edition. url: `http://incompleteideas.net/book/the-book-2nd.html`.

[6] Thomas Fösel, Murphy Yuezhen Niu, Florian Marquardt, and Li Li. "Quantum circuit optimization with deep reinforcement learning" (2021). arXiv:2103.07585.

[7] Zikun Li, Jinjun Peng, Yixuan Mei, Sina Lin, Yi Wu, Oded Padon, and Zhihao Jia. "Quarl: A learning-based quantum circuit optimizer" (2023). arXiv:2307.10120.

[8] Bob Coecke and Ross Duncan. "Interacting quantum observables: categorical algebra and diagrammatics". New Journal of Physics **13**, 043016 (2011).

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms" (2017). arXiv:1707.06347.

[10] Jan Nogué. "Reinforcement Learning based Circuit Compilation via ZX-calculus". Master's thesis. Universitat de Barcelona. (2023). url: `https://diposit.ub.edu/dspace/bitstream/2445/202911/1/Memoria_TFM-JanNogue.pdf`.

[11] Daniel Gottesman. "Theory of fault-tolerant quantum computation". Phys. Rev. A **57**, 127–137 (1998).

[12] Daniel Gottesman. "The heisenberg rep-

resentation of quantum computers" (1998). arXiv:quant-ph/9807006.

[13] Aleks Kissinger and John van de Wetering. "Pyzx: Large scale automated diagrammatic reasoning". Electronic Proceedings in Theoretical Computer Science **318**, 229–241 (2020).

[14] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. "A comprehensive survey on graph neural networks". IEEE Transactions on Neural Networks and Learning Systems **32**, 4–24 (2021).

[15] Bob Coecke and Aleks Kissinger. "Picturing quantum processes: A first course in quantum theory and diagrammatic reasoning". Cambridge University Press. (2017).

[16] John van de Wetering. "Zx-calculus for the working quantum computer scientist" (2020). arXiv:2012.13966.

[17] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. "Graph-theoretic simplification of quantum circuits with the ZX-calculus". Quantum **4**, 279 (2020).

[18] Niel de Beaudrap, Aleks Kissinger, and John van de Wetering. "Circuit extraction for zx-diagrams can be p-hard". In Schloss Dagstuhl – Leibniz-Zentrum für Informatik. (2022). arXiv:2202.09194.

[19] Calum Holker. "Causal flow preserving optimisation of quantum circuits in the zx-calculus" (2024). arXiv:2312.02793.

[20] Korbinian Staudacher, Tobias Guggemos, Wolfgang Gehrke, and Sophia Grundner-Culemann. "Reducing 2-qubit gate count for zx-calculus based quantum circuit optimization". In Quantum Processing and Languages (QPL22). Pages 1–17. (2022). url: https://elib.dlr.de/188470/.

[21] Anton Kotzig. "Eulerian lines in finite 4-valent graphs and their transformations". In *Colloqium on Graph Theory Tihany 1966*Pages pages 219 – 230 (Academic Press, 1968).

[22] André Bouchet. "Graphic presentations of isotropic systems". J. Comb. Theory Ser. A **45**, 58–76 (1987).

[23] Daniel E Browne, Elham Kashefi, Mehdi Mhalla, and Simon Perdrix. "Generalized flow and determinism in measurement-based quantum computation". New Journal of Physics **9**, 250 (2007).

[24] Robert Raussendorf, Daniel Browne, and Hans Briegel. "The one-way quantum computer–a non-network model of quantum computation". Journal of Modern Optics **49**, 1299–1306 (2002).

[25] Miriam Backens, Hector Miller-Bakewell, Giovanni de Felice, Leo Lobski, and John van de Wetering. "There and back again: A circuit extraction tale". Quantum **5**, 421 (2021).

[26] Aleks Kissinger and John van de Wetering. "Reducing the number of non-clifford gates in quantum circuits". Phys. Rev. A **102**, 022406 (2020).

[27] Vincent Danos and Elham Kashefi. "Determinism in the one-way model". Phys. Rev. A **74**, 052310 (2006).

[28] Mehdi Mhalla and Simon Perdrix. "Finding optimal flows efficiently". Page 857–868. Springer Berlin Heidelberg. (2008).

[29] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. "High-dimensional continuous control using generalized advantage estimation" (2018). arXiv:1506.02438.

[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. "Graph attention networks" (2018). arXiv:1710.10903.

[31] Shaked Brody, Uri Alon, and Eran Yahav. "How attentive are graph attention networks?" (2022). arXiv:2105.14491.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "Pytorch: An imperative style, high-performance deep learning library" (2019). arXiv:1912.01703.

[33] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "Openai gym" (2016) arXiv:1606.01540.

[34] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. "Graph

matching networks for learning the similarity of graph structured objects". In International conference on machine learning. Pages 3835–3845. PMLR (2019).

[35] Neil J Ross. code: njross/optimizer.

[36] Yunseong Nam, Neil J. Ross, Yuan Su, Andrew M. Childs, and Dmitri Maslov. "Automated optimization of large quantum circuits with continuous parameters". npj Quantum Information**4** (2018).

[37] Palak Chawla, Shweta, K. R. Swain, Tushti Patel, Renu Bala, Disha Shetty, Kenji Sugisaki, Sudhindu Bikash Mandal, Jordi Riu, Jan Nogué, V. S. Prasannaa, and B. P. Das. "Relativistic variational-quantum-eigensolver calculations of molecular electric dipole moments on quantum hardware". Phys. Rev. A **111**, 022817 (2025).

[38] Palak Chawla, Disha Shetty, Peniel Bertrand Tsemo, Kenji Sugisaki, Jordi Riu, Jan Nogue, Debashis Mukherjee, and V. S. Prasannaa. "Vqe calculations on a nisq era trapped ion quantum computer using a multireference unitary coupled cluster ansatz: application to the beh_2 insertion problem" (2025). arXiv:2504.07037.

[39] Aashna Anil Zade, Kenji Sugisaki, Matthias Werner, Ana Palacios, Jordi Riu, Jan Nogue, Artur Garcia-Saez, Arnau Riera, and V. S. Prasannaa. "Capturing strong correlation effects on a quantum annealer: calculation of avoided crossing in the h_4 molecule using the quantum annealer eigensolver" (2025). arXiv:2412.20464.

[40] Maximilian Nägele and Florian Marquardt. "Optimizing zx-diagrams with deep reinforcement learning" (2023). arXiv:2311.18588.

[41] Francois Charton, Alexandre Krajenbrink, Konstantinos Meichanetzidis, and Richie Yeung. "Teaching small transformers to rewrite ZX diagrams". In The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23. (2023). url: https://openreview.net/forum?id=btQ7Bt1NLF.

[42] Jordi Riu and Jan Nogué. "Code for quantum circuit optimization with rl via zx-calculus". Github Repository (2023). url: https://github.com/qilimanjaro-tech/Circopt-RL-ZXCalc.