Quantum Circuit Synthesis with Deep Reinforcement Learning and Heuristic Search

Ian Turner[1,2], **Forest Agostinelli**[1,2], **Peng Fu**[2]

[1]AI Institute, University of South Carolina, Columbia, South Carolina, USA
[2]Department of Computer Science and Engineering, University of South Carolina, Columbia, South Carolina, USA

Abstract

Quantum circuit synthesis is the process of implementing a quantum algorithm with a given discrete gate set. In this paper, we first show how this problem can be posed as a pathfinding problem. Next, we use DeepCubeA to learn a heuristic function for quantum circuit synthesis for one to three qubit circuits with deep reinforcement learning and solve problem instances with batch weighed A* search. We compare our approach against several other state-of-the-art quantum circuit synthesis algorithms and show that are our approach is competitive.

Introduction

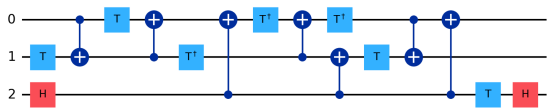

Figure 1: Implementation of the Toffoli operator in the Clifford+T gate set

The realization of quantum algorithms on real-world quantum computers is accomplished through quantum compiling (Maronese et al. 2022), also known as quantum circuit synthesis. Given a quantum algorithm, quantum compiling finds a quantum circuit that implements that algorithm. A quantum algorithm can be represented as a matrix of size $2^n \times 2^n$, with n being the number of qubits. A quantum circuit (see Figure 1) is a sequence of quantum gates that perform operations on one or more qubits, where a qubit is an irreducible unit of quantum information. Given a quantum circuit, the matrix that represents the algorithm it implements is obtained by starting with the identity matrix and changing its entries according to matrix multiplications determined by the gates used in the circuit. As a result, quantum compiling can be posed as a pathfinding problem, where the start state is the identity matrix, the goal is a given quantum algorithm, the transitions are quantum gates, and the transition costs are determined by a combination of quantum gate execution time and the noise the gate introduces.

Finding quantum circuits that implement a given algorithm is a non-trivial task that can sometimes take state-of-the-art methods several hours, even for single qubit operators (Paradis et al. 2024). Given that we can pose this as a pathfinding problem, we build on the DeepCubeA algorithm (Agostinelli et al. 2019) and hindsight experience replay (HER) (Andrychowicz et al. 2017) to use deep reinforcement learning to learn a heuristic function that maps a given quantum circuit and goal quantum algorithm to an estimate of the cost of a shortest path (i.e. "cost-to-go") from the given quantum circuit to a quantum circuit that implements the given goal quantum algorithm. Previous work that built on DeepCubeA to perform quantum compiling used the reversibility of quantum gates to set the starting state to be the quantum algorithm and the goal to be the identity matrix (Zhang et al. 2020; Bao and Hartnett 2024; Chen et al. 2024). Our work distinguishes itself from this previous work since the starting state is the identity matrix and the goal is given as the quantum algorithm. This distinction then allows our approach to be modified for the specification of goals as a set of goal states through partial specification of goal matrix entries or through abstraction using formal logic. This may be necessary if certain properties of the algorithm (i.e. entries in the matrix) are known while other parts are not known or not relevant. These modifications that allow for this are already present in existing literature (Agostinelli, Panta, and Khandelwal 2024; Agostinelli 2025) and can be applied to the work presented in the paper.

Background

Quantum Computing

Qubits & Unitary Operators A qubit is the fundamental unit of information in quantum computing. We can represent any system of n qubits as a unit vector $|\psi\rangle \in \mathbb{C}^{2^n}$. The 0 and 1 states for a single qubit can be represented as

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{1}$$

and a general one-qubit quantum state can be represented by $\alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. It is important to note that if two quantum state vectors differ only by a complex phase (meaning $|\psi\rangle = e^{i\theta}|\phi\rangle$ for two state vectors $|\psi\rangle$ and $|\phi\rangle$), then the two states cannot be distinguished

in any way by measurement. For this reason, it is common to disregard global phases in calculations or modify them as needed, and these states are considered equivalent (Nielsen and Chuang 2010).

Quantum gates transform quantum state vectors into new state vectors. Disregarding measurement operations, a quantum gate can be represented by a unitary matrix $U \in \mathbb{C}^{2^n \times 2^n}$. A matrix U is unitary if $U^\dagger U = UU^\dagger = I$, where $U^\dagger$ is the conjugate transpose of U. As a consequence of the global phase invariance property of quantum state vectors, two gates whose unitaries differ only by a global phase can be considered to be equivalent.

There are many methods of calculating the 'distance' between two unitary operators. In this paper we use the Hilbert-Schmidt distance function

$$d(U,V) = \sqrt{1 - \frac{1}{4^n}|Tr(U^\dagger V)|^2} \tag{2}$$

This method has the advantage that if two unitaries A and B differ by only a global phase, then $d(A,B) = 0$.

Clifford+T Gate Set Any unitary can be approximated within an arbitrary tolerance ϵ by a finite gate set if that gate set is universal (Nielsen and Chuang 2010). The most commonly studied gate set is the Clifford+T set, as it is likely to be the instruction set of near-term fault-tolerant quantum computers. It is a union of the Clifford gate set and the T gate. The Clifford set can be generated (via matrix multiplication and tensor product) by the three Pauli matrices

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \; Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \; Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{3}$$

in combination with the Hadamard and S gates, denoted

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \; S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \tag{4}$$

Note that this is not the only generating gate set for the Clifford operators. Since the Clifford gate set is not universal, we must add the T gate

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{5}$$

This is a discrete gate set, in contrast to gates like the $R_z(\theta)$ gate (where θ is in the set of real numbers), which has the form

$$R_z(\theta) = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix} \tag{6}$$

These gates are extremely common in quantum algorithms, so finding efficient Clifford+T decompositions of these gates is crucial for scaling quantum algorithms to larger machines.

Unitary Synthesis Given that discrete gate sets such as Clifford+T are likely to be the basis for near-term fault-tolerant quantum computing, methods that can synthesize arbitrary unitaries from these discrete gate sets are needed. The problem of unitary synthesis is this: given a unitary specification U, find a sequence of gates $\{g_1, ..., g_n\}$ chosen from a discrete gate set that implements a unitary $V = g_n \times ... \times g_1$ such that $d(V,U) \leq \epsilon$, where ϵ is the 'tolerance' value of the compilation. When $\epsilon > 0$, we classify the problem as approximate synthesis, and when $\epsilon = 0$, we classify it as exact synthesis, although in reality numerical errors will occur, so we choose some $\epsilon \ll 1$ (in this work we chose $\epsilon = 10^{-6}$). For small operators the synthesized circuit can then be evaluated for exact accuracy using by-hand calculations.

Pathfinding

Pathfinding is the process of finding a sequence of actions that forms a path between a given start state and a given goal, where a goal is a set of states considered goal states. A pathfinding domain is defined by a weighted directed graph, where nodes represent states, edges represent transitions between states, and weights represent transition costs (Pohl 1970). The transitions can be thought of as resulting from a set of actions, $\mathcal{A}$, and the transition function function, T, returns state, s', given state, s, and action, a, if and only if there is an edge between state s and s' for some action, a (i.e. $s' = T(s,a)$). The transition cost function, c, returns the cost of taking action, a, in states, s (i.e. $c(s,a)$). Given a domain, a pathfinding problem instance is defined by a start state and a goal. Given a path that is a solution to a problem instance, the path cost is the sum of transition costs. In this work, the states are matrices representing quantum algorithms, transitions are the application of quantum gates, which produce new matrices, and transition costs are 1 for all gates. In future work, we will consider assigning a higher transition cost to gates, such as the T gate, which add more execution time or noise to the circuit.

Heuristic Search A* search (Hart, Nilsson, and Raphael 1968) is a search algorithm designed to find a path between two nodes in a weighted directed graph. Similar to other algorithms like depth-first search and Djikstra's algorithm, A* keeps a priority queue of nodes, ordered from least to greatest according to the function

$$f(n) = g(n) + h(n) \tag{7}$$

where $g(n)$ is the path cost (sum of all transition costs taken to reach node n and $h(n)$ is the heuristic function value of state associated with n.

While searching each node keeps track of the path cost and heuristic function value as well as a connection to its parent node and the action taken to generate the state from its parent. The A* search process is begun when a single node with no parents representing the start state is added to the queue, and is ended once a node associated with a goal state is selected for expansion or there are no more nodes to expand.

In practice, a significant speedup can be achieved with heuristic functions represented by neural networks by batching the expansion steps together and exploiting the parallelism of graphics processing units (GPUs), as well as modifying the cost function to be $f(n) = \lambda g(n) + h(n)$, where $\lambda \in [0,1]$ can be adjusted to place less important on the path cost of a state and more on the heuristic value. This batched and weighted version of A* search is referred to as batch weighted A* search (BWAS).

DeepCubeA

DeepCubeA (Agostinelli et al. 2019) is an algorithm that learns a domain-specific heuristic function represented as a deep neural network (DNN) (Schmidhuber 2015; LeCun, Bengio, and Hinton 2015) using deep reinforcement learning (Sutton and Barto 2018), which is then used with BWAS to solve problem instances. While DeepCubeA originally assumed the goal was known before training and took transitions in reverse from the goal, extensions have built on hindsight experience replay (HER) (Andrychowicz et al. 2017) to learn heuristic functions that generalize over both states and goals (Agostinelli, Panta, and Khandelwal 2024; Agostinelli and Soltani 2024).

Learning the Heuristic Function The heuristic function is learned with approximate value iteration (Puterman and Shin 1978; Bertsekas and Tsitsiklis 1996), which iteratively trains a model to approximate a Bellman update for a set of problem instances. The Bellman update, in the context of pathfinding (shown in Equation 8) uses the Bellman optimality equation as an update rule. Approximate value iteration can be used to train a DNN, h_θ, with parameters θ, with gradient descent using the loss function in Equation 9, where N is the batch size. The heuristic function used to calculate the Bellman update in Equation 8 is h_{θ^-}, where θ^- is the parameters of the target network (Mnih et al. 2015) which are periodically updated to θ. Approximate value iteration with DNNs is referred to as deep approximate value iteration (DAVI).

$$h'(s,g) = \begin{cases} 0, & \text{if } s \in g, \\ \min\limits_{a \in \mathcal{A}} c(s,a) + h(T(s,a),g), & \text{otherwise.} \end{cases} \tag{8}$$

$$L(\theta) = \frac{1}{N} \sum_{i}^{N} \left(h'(s_i, g_i) - h_\theta(s_i, g_i) \right)^2 \tag{9}$$

We build on hindsight experience replay (HER) (Andrychowicz et al. 2017) to generate start state and goal pairs without having to start from a predetermined goal and take actions in reverse. We accomplish this by first starting with the identity matrix and taking random actions to generate a start state. From this generated start state, we then take random actions to generate a goal state.

Related Work

Many methods for unitary synthesis have been explored. The programs **synthetiq** (Paradis et al. 2024) and **trasyn** (Hao, Xu, and Tannu 2025) use random gate sampling search guided by the unitary distance function. The program gridsynth (Ross and Selinger 2014) uses advanced domain-specific knowledge to analytically synthesize optimal circuits, but is limited to only one-qubit $R_z(\theta)$ gates. These methods place higher importance on minimizing T count at the expense of longer Clifford gate counts, given that T gates are regarded as more hardware intensive.

Deep reinforcement learning and heuristic search has been used to synthesize quantum circuits. Zhang et al. (2020) build on DeepCubeA to learn a heuristic function for topological quantum compilation. (Bao and Hartnett 2024) learn a heuristic function that is then used with beam search for Clifford gates. (Chen et al. 2024) use deep Q-learning (Watkins and Dayan 1992; Mnih et al. 2015) and Q* search (Agostinelli et al. 2024) to perform single and multi-qubit synthesis. Rietsch et al. (2024) explores both policy gradient methods and Q-learning. (Weiden et al. 2025) explores modifications to the neural networks and training processes used in Clifford+T synthesis. While these previous approaches assume a fixed goal of the identity matrix and use reverse quantum gates to find a path from a specified algorithm to the identity, we instead find a path from the identity to the specified algorithm using the given gate set. This then allows future iterations of our work to include the specification of sets of goal states through partial goal specification or abstraction through formal logic (Agostinelli, Panta, and Khandelwal 2024; Agostinelli 2025), offering greater flexibility to practitioners and reducing the knowledge required to specify an algorithm.

Approach

State Representation

States are represented as unitary matrices. These matrices are the product of the matrices of all the actions taken to get to the state, multiplied together. We take advantage of the fact that unitary matrices are invertible, and multiply the inverse of the state matrix with the goal matrix when giving it as input to the neural network. We are allowed to do this because the problem of finding a matrix M such that $MS = G$ is the same as finding a matrix M such that $M = GS^{-1}$ (here S is the matrix of the current state, G is the matrix of the goal state, and M is the matrix representing the actions needed to go from the state to the goal). This gives us the overall transformation from state to goal, which is also a unitary matrix.

When inputting the unitary matrices into the DNN we utilize the generalized Euler angle parametrization of $U(n)$ (Diaconis and Forrester 2017), which maps a unitary matrix to a vector of n^2 angles, called 'Euler angles.' Another degree of freedom can be removed by removing the global phase, resulting in an overall function $\eta : U(n) \rightarrow [0, 2\pi)^{n^2-1}$. This reduces the $2n^2$ real degrees of freedom (resulting from the n^2 complex entries of the matrix) of the unitary group into $n^2 - 1$ real degrees of freedom , removing all redundancies and essentially imposing a 'coordinate system' on the group. Other works have discussed the use of Euler angles to represent unitary operators (Alam, Berthusen, and Orth 2023), and pointed out that they have some downsides when compared to the quaternion representation. However, quaternions can only be used to represent $SU(2)$ (the case of one-qubit unitaries up to a global phase), and do not generalize to multiple-qubit systems.

We also use Neural Radiance Field Encoding (NeRF) (Mildenhall et al. 2021) to represent the matrices to the DNN, which was originally developed for image processing, but has been shown to improve neural network performance when dealing with unitary matrices (Weiden et al. 2025). NeRF encoding can be described as a function $\gamma : \mathbb{R} \rightarrow \mathbb{R}^{2L}$

that maps real numbers to a vector of $2L$ real numbers, where L is an arbitrary parameter. The function is defined as

$$\gamma(x) = \begin{bmatrix} \cos(2^0\pi x) \\ \sin(2^0\pi x) \\ \vdots \\ \cos(2^{L-1}\pi x) \\ \sin(2^{L-1}\pi x) \end{bmatrix} \tag{10}$$

It ensures the input is between -1 and 1, which has been shown to improve the performance of neural networks.

Training

We use HER to generate training examples and, additionally, attempt to solve problem instances with a greedy policy using the target network and add states seen during solution attempts to the training set. The heuristic function is then trained with DAVI. After using HER to generate goal states, we apply perturbations to each goal matrix U to generate matrices $\tilde{U}$ such that $d(U, \tilde{U}) \leq \epsilon$, similar to the training process in (Weiden et al. 2025). This has been shown to make the training process more robust to approximate synthesis.

After each update, we test our value function using a greedy policy. This policy can be defined as

$$\pi(s) = \underset{a \in \mathcal{A}}{\operatorname{argmin}}(c(s,a) + h_\theta(T(s,a), s_g)) \tag{11}$$

where s_g is the goal state. The greedy policy simply selects the state which minimizes the value of the heuristic function plus the transition cost for all possible next states. We record the percentage of test states solved by the greedy policy after each iteration, and update our target network parameters θ^- only when this percentage has surpassed the previous best. A graph of this percentages after each training update is shown in figure 2 for one-qubit models with and without Euler angle encoding, NeRF, and perturbations.

Testing & Verification

For single qubit synthesis we use parametrized Z rotation gates, denoted $R_z(\theta)$ as a test dataset. We run A* search on each of these gates and compare the resulting circuits with those generated using **trasyn**. For multi-qubit synthesis we run on a small variety of commonly-used operators, mostly chosen from the benchmarks used for **Synthetiq**. We save the circuits in the OpenQASM format (Cross et al. 2022), and import them into Qiskit to check their closeness to the original operators.

Experiments

Single-Qubit Approximate Synthesis

For single-qubit synthesis we trained models with and without NeRF, perturbations, and Euler angle encoding. We found the best performance when using a combination of Euler angle encoding, perturbations, and NeRF with $L = 15$. Most of the performance boost was attributable to NeRF, although Euler angle encoding and perturbations seem to lead to more stable training and slightly better search results.

We trained models for epsilon values of 0.01 and 0.005. We then ran A* search with a batch size of 200 and a path weight of 0.2 and compared the results with **trasyn**. Both A* search and **trasyn** were able to find a circuit in all cases, however A* search often finds circuits with an lower overall gate count, but the same T-count as **trasyn**. This is likely because **trasyn** places its emphasis on optimizing T-count, since T gates are considered to be much more hardware-intensive to implement, however the shortest circuit is typically also a circuit that uses the minimal amount of T gates required to approximate a target unitary. Overall, A* search still finds circuits faster than **trasyn** in most cases, and seems to scale better to the lower epsilon value than **trasyn**, which increases in search time drastically when going from epsilon of 0.01 to 0.005. These results are shown in table 1.

Multi-Qubit Exact Synthesis

For multi-qubit synthesis we trained models both with and without NeRF, but did not see any distinguishable difference between the performance of the two. We also trained models with and without Euler angle encoding, but the models with Euler angle encoding performed significantly worse than those without it.

We found the best performance for A* search using a batch size of 10 and a path cost of 1.0. We compared our results to **synthetiq**, which uses simulated annealing to synthesize circuits, where the energy function is the Hilbert-Schmidt distance function. We ran **synthetiq** on 48 cores, generating 100 circuits for every operator and picking the best one. The results for synthesis of several commonly-used operators are in Table 2. The results show that our approach performs comparable to that of **synthetiq** on all metrics. For a specific operator, CCH, our approach finds a solution 186x faster and with fewer gates while for CCX, **synthetiq** finds a solution 10x faster and with fewer gates.

Discussion and Future Work

This work only considers unitary operations, but some quantum computers now support dynamic circuits, which involve measurements during the circuit and further operations conditioned on the measurement outcomes. Previous work has shown that repeat-until-success circuits can be used to create more optimal fault-tolerant gates (Bocharov, Roetteler, and Svore 2015). 'Magic' state distillation (Bravyi and Kitaev 2005) has also been explored as a potential way of implementing non-Clifford gates fault-tolerantly. The measurement-based quantum computing paradigm even offers an alternative to the unitary quantum circuit model, where the same 'resource state' is prepared every time, and computation is carried out exclusively by measurements (Briegel et al. 2009). Future work could involve expanding the search space to include those more exotic circuits.

Our approach gives the desired quantum algorithm as a goal to the heuristic function. There may be cases where only partial information of the properties the quantum algorithm should have are known. In these cases, this could correspond to not knowing all entries in the matrix that represents the algorithm. Work on generalizing over goals can be

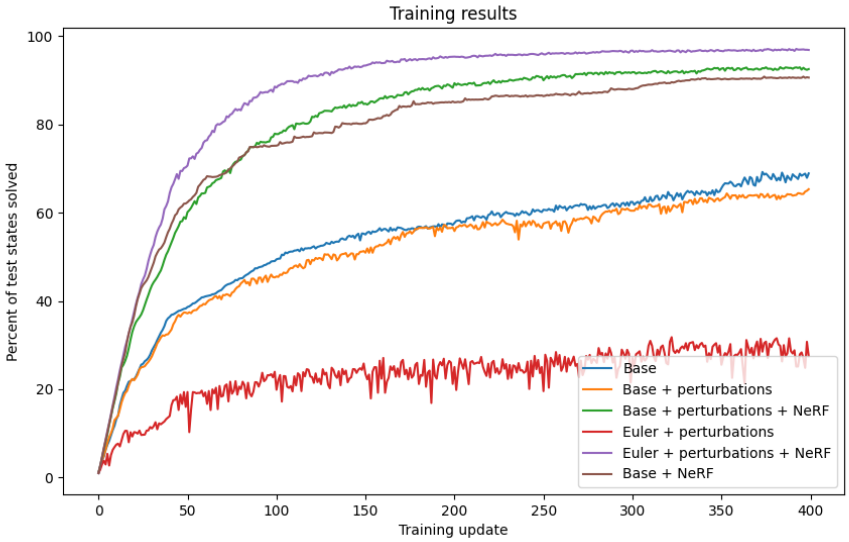

Figure 2: Percent of test states solved using a greedy policy after each training update for different encoding methods both with and without perturbations added for single-qubit approximate synthesis. Here 'base' indicates inputting the matrix into the DNN without Euler angle encoding.

k	ϵ	Gate Count		T-Count		Time	
		A*	Trasyn	A*	Trasyn	A*	Trasyn
4	0.01	**43**	45	18	18	7.225	19.488
5	0.01	**39**	41	17	17	6.585	9.909
6	0.01	**42**	43	18	18	6.883	19.636
7	0.01	**38**	39	15	15	6.398	2.286
4	0.005	45	45	18	18	8.628	22.599
5	0.005	**45**	49	20	20	40.326	86.470
6	0.005	**45**	47	19	19	6.390	45.342
7	0.005	**45**	47	19	19	6.784	44.147

Table 1: A* search vs. Trasyn for single-qubit approximate synthesis of $R_z(2\pi/2^k)$ gates

Target	Qubits	Gate-count		T-count		Time	
		A*	Synthetiq	A*	Synthetiq	A*	Synthetiq
CH	2	7	7	2	2	0.741	0.342
CZ	2	3	3	0	0	0.518	0.252
$CR_z(\pi/2)$	2	4	4	2	2	1.369	0.542
CR_2	2	5	5	3	3	2.316	0.201
CCH	3	**23**	25	9	9	7.650	1304.018
CCZ	3	**13**	16	7	7	24.279	5.718
Fredkin (C-SWAP)	3	19	19	7	7	75.574	19.025
Toffoli (CCX)	3	16	**15**	7	7	74.052	7.490

Table 2: A* search vs. Synthetiq for multi-qubit exact synthesis

used to train a heuristic function that can be given a partially specified algorithm by having a special indicator for entries that are not known or through abstractions based on formal logic (Agostinelli, Panta, and Khandelwal 2024; Agostinelli 2025).

Conclusion

Efficient unitary synthesis will be crucial to fault-tolerant quantum computing in the near future, allowing programmers to translate their high-level algorithm descriptions to low-level implementations on quantum hardware. We extend DeepCubeA and HER to learn a domain-specific heuristic to estimate cost-to-go from a given algorithm to a given goal algorithm for the Clifford+T gate set. We then use this heuristic function in combination with BWAS to find circuits that implement the given goal algorithm. We compare the results of BWAS with two state-of-the-art unitary synthesis programs. Future work could achieve improvements by finding a better encoding for unitary matrices and extending the algorithm to allow for synthesis of partial specifications and dynamic circuits.

References

Agostinelli, F. 2025. A Conflict-Driven Approach for Reaching Goals Specified with Negation as Failure. In *Proceedings of the International Symposium on Combinatorial Search*, volume 18, 2–10.

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8): 356–363.

Agostinelli, F.; Panta, R.; and Khandelwal, V. 2024. Specifying goals to deep neural networks with answer set programming. In *34th International Conference on Automated Planning and Scheduling*.

Agostinelli, F.; Shperberg, S. S.; Shmakov, A.; McAleer, S.; Fox, R.; and Baldi, P. 2024. Q* search: Heuristic search with deep q-networks. In *ICAPS Workshop on Bridging the Gap between AI Planning and Reinforcement Learning*.

Agostinelli, F.; and Soltani, M. 2024. Learning discrete world models for heuristic search. In *Reinforcement Learning Conference*.

Alam, M. S.; Berthusen, N. F.; and Orth, P. P. 2023. Quantum logic gate synthesis as a Markov decision process. *npj Quantum Information*, 9(1): 108.

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Pieter Abbeel, O.; and Zaremba, W. 2017. Hindsight experience replay. *Advances in neural information processing systems*, 30.

Bao, N.; and Hartnett, G. S. 2024. Twisty-puzzle-inspired approach to Clifford synthesis. *Physical Review A*, 109(3): 032409.

Bertsekas, D. P.; and Tsitsiklis, J. N. 1996. *Neuro-dynamic programming*. Athena Scientific. ISBN 1-886529-10-8.

Bocharov, A.; Roetteler, M.; and Svore, K. M. 2015. Efficient synthesis of universal repeat-until-success quantum circuits. *Physical review letters*, 114(8): 080502.

Bravyi, S.; and Kitaev, A. 2005. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Physical Review A—Atomic, Molecular, and Optical Physics*, 71(2): 022316.

Briegel, H. J.; Browne, D. E.; Dür, W.; Raussendorf, R.; and Van den Nest, M. 2009. Measurement-based quantum computation. *Nature Physics*, 5(1): 19–26.

Chen, Q.; Du, Y.; Jiao, Y.; Lu, X.; Wu, X.; and Zhao, Q. 2024. Efficient and practical quantum compiler towards multi-qubit systems with deep reinforcement learning. *Quantum Science and Technology*, 9(4): 045002.

Cross, A.; Javadi-Abhari, A.; Alexander, T.; De Beaudrap, N.; Bishop, L. S.; Heidel, S.; Ryan, C. A.; Sivarajah, P.; Smolin, J.; Gambetta, J. M.; et al. 2022. OpenQASM 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing*, 3(3): 1–50.

Diaconis, P.; and Forrester, P. J. 2017. Hurwitz and the origins of random matrix theory in mathematics. *Random Matrices: Theory and Applications*, 6(01): 1730001.

Hao, T.; Xu, A.; and Tannu, S. 2025. Reducing T Gates with Unitary Synthesis. *arXiv preprint arXiv:2503.15843*.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *nature*, 521(7553): 436.

Maronese, M.; Moro, L.; Rocutto, L.; and Prati, E. 2022. Quantum compiling. In *Quantum Computing Environments*, 39–74. Springer.

Mildenhall, B.; Srinivasan, P. P.; Tancik, M.; Barron, J. T.; Ramamoorthi, R.; and Ng, R. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1): 99–106.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Nielsen, M. A.; and Chuang, I. L. 2010. *Quantum computation and quantum information*. Cambridge university press.

Paradis, A.; Dekoninck, J.; Bichsel, B.; and Vechev, M. 2024. Synthetiq: Fast and Versatile Quantum Circuit Synthesis. volume 8.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4): 193–204.

Puterman, M. L.; and Shin, M. C. 1978. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24(11): 1127–1137.

Rietsch, S.; Dubey, A. Y.; Ufrecht, C.; Periyasamy, M.; Plinge, A.; Mutschler, C.; and Scherer, D. D. 2024. Unitary synthesis of clifford+ t circuits with reinforcement learning. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, 824–835. IEEE.

Schmidhuber, J. 2015. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Watkins, C. J.; and Dayan, P. 1992. Q-learning. *Machine learning*, 8(3-4): 279–292.

Weiden, M.; Kalloor, J.; Kubiatowicz, J.; and Iancu, C. 2025. Making Neural Networks More Suitable for Approximate Clifford+ T Circuit Synthesis. *arXiv preprint arXiv:2504.15990*.

Zhang, Y.-H.; Zheng, P.-L.; Zhang, Y.; and Deng, D.-L. 2020. Topological quantum compiling with reinforcement learning. *Physical Review Letters*, 125(17): 170501.