

Quantum circuit optimization with AlphaTensor

Received: 8 March 2024

Accepted: 20 January 2025

Published online: 20 March 2025

 Check for updates

Francisco J. R. Ruiz^{1,5}✉, Tuomas Laakkonen^{2,5}, Johannes Bausch¹, Matej Balog¹, Mohammdamin Barekatin¹, Francisco J. H. Heras¹, Alexander Novikov¹, Nathan Fitzpatrick³, Bernardino Romera-Paredes¹, John van de Wetering⁴, Alhussein Fawzi¹, Konstantinos Meichanetzidis² & Pushmeet Kohli¹

A key challenge in realizing fault-tolerant quantum computers is circuit optimization. Focusing on the most expensive gates in fault-tolerant quantum computation (namely, the T gates), we address the problem of T-count optimization, that is, minimizing the number of T gates needed to implement a given circuit. To achieve this, we develop AlphaTensor-Quantum, a method based on deep reinforcement learning that exploits the relationship between optimizing the T-count and tensor decomposition. Unlike existing methods for T-count optimization, AlphaTensor-Quantum can incorporate domain-specific knowledge about quantum computation and leverage gadgets, which substantially reduces the T-count of the optimized circuits. AlphaTensor-Quantum outperforms the existing methods for T-count optimization on a set of arithmetic benchmarks (even when compared without using gadgets). Remarkably, it discovers an efficient algorithm akin to Karatsuba's method for multiplication in finite fields. AlphaTensor-Quantum also finds the best human-designed solutions for relevant arithmetic computations used in Shor's algorithm and for quantum chemistry simulation, thus demonstrating that it can save hundreds of hours of research by optimizing relevant quantum circuits in a fully automated way.

Quantum computation^{1,2} presents a fundamentally new approach to solving computational problems across diverse fields, ranging from cryptography³ and drug discovery⁴ to materials science and high-energy physics⁵. To implement any quantum algorithm, quantum computers require two types of quantum gates: Clifford, for example, Hadamard or controlled NOT (CNOT), and non-Clifford gates, for example, T gates. A quantum circuit using Clifford gates exclusively can be simulated on a classical computer efficiently, that is, in polynomial time (Gottesman–Knill theorem⁶). Therefore, to achieve universality, non-Clifford gates are needed. However, non-Clifford gates have a substantial impact on the runtime and resource cost of

quantum circuits^{7,8}, because fault-tolerant quantum computation requires error-correction schemes that make it impossible to implement a universal gate set transversally^{9–11}, and non-Clifford gates can only be implemented using magic states⁷. Magic states are quantum states that ‘encode’ gates; this can be understood as applying the gates in advance, resulting in magic states that can be consumed later in the computation whenever those gates are needed. However, producing high-quality magic states is a process with high space and time overhead. For example, the cost of the T gate is approximately two orders of magnitude larger than the cost of a CNOT operation between two adjacent qubits¹². Despite recent progress to mitigate that issue¹³, the

¹Google DeepMind, London, UK. ²Quantinuum, Oxford, UK. ³Quantinuum, Cambridge, UK. ⁴Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands. ⁵These authors contributed equally: Francisco J. R. Ruiz, Tuomas Laakkonen. ✉e-mail: franruiz@google.com

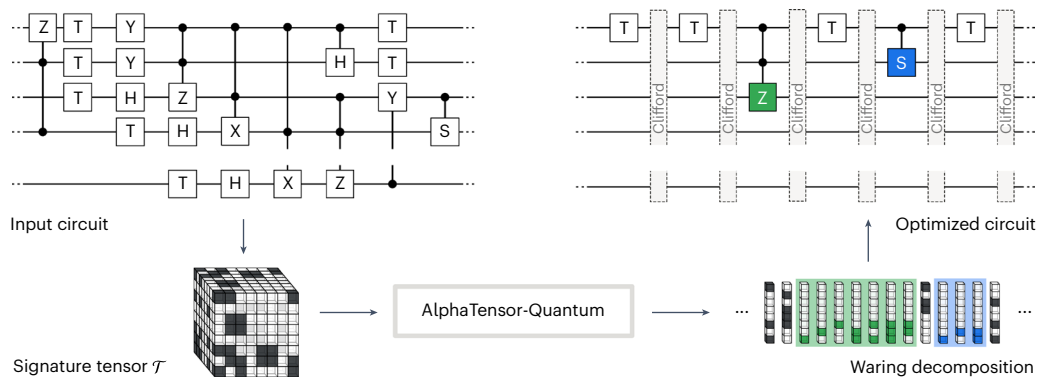


Fig. 1 | Pipeline of AlphaTensor-Quantum. We first extract the non-Clifford components of an input circuit and represent them as a symmetric signature tensor \mathcal{T} , a binary tensor depicted as a cube of solid and transparent blocks. We then use AlphaTensor-Quantum to find a low-rank Waring decomposition of that tensor, that is, a set of factors (displayed as columns of blocks) that can be

mapped back into an optimized quantum circuit with reduced T-count. In general, there is a one-to-one correspondence between factors and T gates, but AlphaTensor-Quantum can also group factors into gadgets (highlighted in green and blue)—constructions that can be equivalently implemented with fewer T gates than the sum of the factors.

cost of fault-tolerant quantum algorithms remains dominated by the cost of implementing the non-Clifford gates.

In this paper, we focus on the problem of T-count optimization—minimizing the number of T gates of quantum algorithms. T-count optimization is an NP-hard problem¹⁴ that has been addressed using different approaches^{15–24}. Following those, we consider the T-count as the sole metric of complexity of a quantum circuit.

With that aim, we first transform the problem into an instance of tensor decomposition, exploiting the relationship between T-count optimization and finding the symmetric tensor rank^{17,19}. We develop AlphaTensor-Quantum, an extension of AlphaTensor²⁵, a method that finds low-rank tensor decompositions using deep reinforcement learning (RL). Numerous studies have recently applied RL to optimize quantum circuits^{26–33}, although they do not specifically tackle T-count optimization. AlphaTensor-Quantum tackles the problem from the tensor decomposition point of view and can find algorithm implementations with low T-count. The approach is illustrated in Fig. 1.

AlphaTensor-Quantum addresses three main challenges that go beyond the capabilities of AlphaTensor²⁵ when applied to this problem. First, it optimizes the symmetric (rather than the standard) tensor rank; this is achieved by modifying the RL environment and actions to provide symmetric (Waring) decompositions of the tensor, which has the beneficial side effect of reducing the action search space. Second, AlphaTensor-Quantum scales up to large tensor sizes, which is a requirement as the size of the tensor corresponds directly to the number of qubits in the circuit to be optimized; this is achieved by a neural network architecture featuring symmetrization layers. Third, AlphaTensor-Quantum leverages domain knowledge that falls outside of the tensor decomposition framework; this is achieved by incorporating gadgets (constructions that can save T gates by using auxiliary ancilla qubits) through an efficient procedure embedded in the RL environment.

We demonstrate that AlphaTensor-Quantum is a powerful method for finding efficient quantum circuits. On a benchmark of arithmetic primitives, it outperforms all existing methods for T-count optimization, especially when allowed to leverage domain knowledge. For multiplication in finite fields, an operation with application in cryptography³⁴, AlphaTensor-Quantum finds an efficient quantum algorithm with the same complexity as the classical Karatsuba method³⁵. This is the most efficient quantum algorithm for multiplication on finite fields reported so far (naive translations of classical algorithms introduce overhead^{36,37} due to the reversible nature of quantum computations). We also optimize quantum primitives for other relevant problems, ranging from arithmetic computations used, for example, in Shor's algorithm³⁸, to Hamiltonian simulation in quantum chemistry, for example, iron–molybdenum cofactor (FeMoco) simulation^{39,40}.

AlphaTensor-Quantum recovers the best-known hand-designed solutions, demonstrating that it can effectively optimize circuits of interest in a fully automated way. We envision that this approach can accelerate discoveries in quantum computation as it saves the numerous hours of research invested in the design of optimized circuits.

AlphaTensor-Quantum can effectively exploit the domain knowledge (provided in the form of gadgets with state-of-the-art magic-state factories¹²), finding constructions with lower T-count. Because of its flexibility, AlphaTensor-Quantum can be readily extended in multiple ways, for example, by considering complexity metrics other than the T-count such as the cost of two-qubit Clifford gates or the qubit topology, by allowing circuit approximations, or by incorporating new domain knowledge. We expect that AlphaTensor-Quantum will become instrumental in automatic circuit optimization with new advancements in quantum computing.

AlphaTensor-Quantum for T-count optimization

Quantum gates

Quantum computers work with qubits, representable as L2-normalized vectors in \mathbb{C}^2 , typically denoted with the bra-ket notation as $|\psi\rangle$, with the basis vectors being $|0\rangle$ and $|1\rangle$. An N -qubit state resides in \mathbb{C}^{2^N} . Quantum circuits, implemented by quantum gates, perform linear normalization-preserving operations over quantum states, thus they can be represented by unitary matrices.

We focus on the Clifford+T gate set because it is the leading candidate for fault-tolerant quantum computing and it is approximately universal, meaning that any unitary matrix can be approximated arbitrarily well by a circuit consisting of these gates. Specifically, we consider the Hadamard (H) and CNOT, which are Clifford gates, and the T gate, given by the unitary matrices

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}, \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

$$\text{and} \quad \text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

We also define the following Clifford gates:

$$S = T^2, \quad Z = S^2, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \text{and} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}.$$

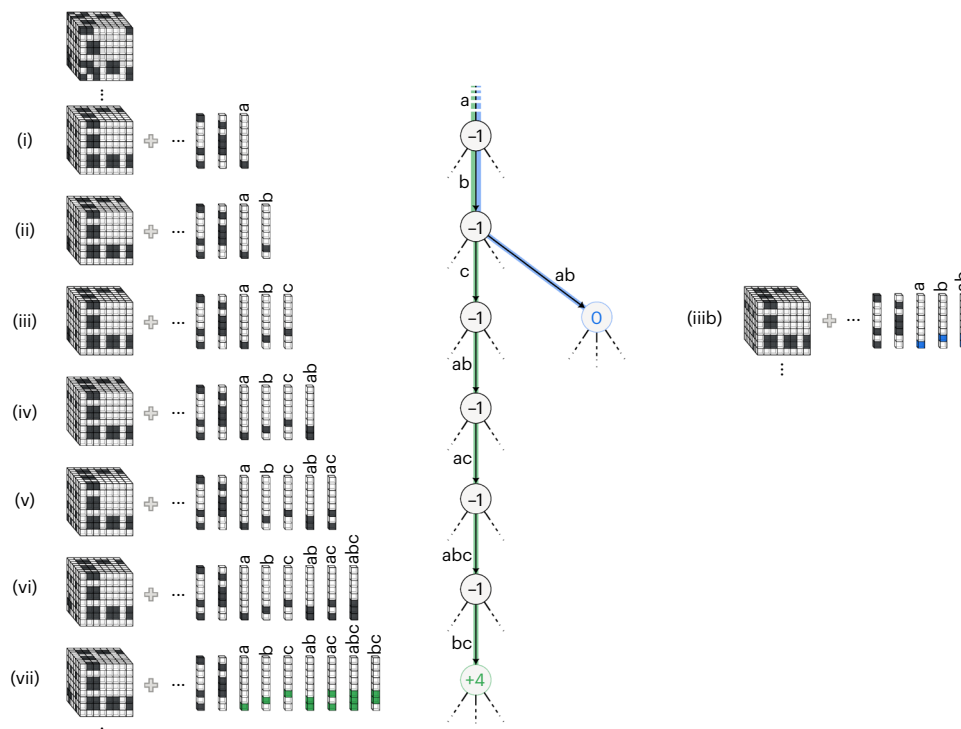


Fig. 2 | Illustration of gadgetization in TensorGame. Nodes in the tree correspond to states, and the number in each node is the immediate reward associated with the action leading to that node. In state (i), the last played action is labelled 'a', and state (ii) is reached after playing another action (labelled 'b'). From state (ii), playing action 'c' leads to state (iii), incurring an additional -1 reward. If action 'ab' is played instead from state (ii), the reward leading to state (iiib) is 0 because the move completes a CS gadget (blue path). Similarly, the sequence of moves in the green path completes a Toffoli gadget, and thus the

immediate reward in state (vii) is +4 so that the last 7 actions jointly receive a reward of -2 (see Methods for details on the gadgetization patterns). With its ability to plan, from state (i) the agent may decide to play actions in the green or blue paths and benefit from the adjusted rewards, or to play other actions that are not part of a gadget (black dashed paths). In this way, AlphaTensor-Quantum can automatically find trade-offs between playing actions that are part of a gadget and actions that are not.

Except for the CNOT, these gates act on single qubits. The T, S and Z are phase-rotation gates, for example, $T|x\rangle = e^{i\frac{\pi}{4}x}|x\rangle$ for a quantum state $|x\rangle$ with $x \in \{0, 1\}$. The CNOT acts on two qubits; it flips the second qubit if the first qubit is $|1\rangle$, that is, $\text{CNOT}|x, y\rangle = |x, x \oplus y\rangle$, where ' \oplus ' denotes the XOR operation. Controlled gates like the CNOT, summarized in Extended Data Fig. 1, enable quantum entanglement. For instance, the Toffoli gate (controlled CNOT) acts as $\text{Tof}|x, y, z\rangle = |x, y, z \oplus (xy)\rangle$. Composition of quantum gates is achieved via matrix multiplication (serial) and Kronecker product (parallel).

Signature tensor

Given an N -qubit circuit composed of CNOT and T gates alone, we can encode the information about the non-Clifford components into a symmetric signature tensor $\mathcal{T} \in \{0, 1\}^{N \times N \times N}$ (see Extended Data Fig. 2 for an example). Two CNOT+T circuits implement the same unitary matrix, up to Clifford gates, if and only if they have the same signature tensor^{17,19} (Methods).

To optimize the T-count, we first find the circuit representation in terms of the signature tensor \mathcal{T} . When the circuit has gates other than CNOT and T gates, to obtain \mathcal{T} we first split the circuit into two parts^{17,41}—a diagonal part that contains CNOT and T gates only, and a non-diagonal part that consists of Clifford gates (possibly containing CNOT gates too, to undo the effects of the diagonalization). See Fig. 3 for an example and Supplementary Section C.1 for details.

Afterwards, we obtain \mathcal{T} by first mapping the r th T gate to a vector $\mathbf{u}^{(r)} \in \{0, 1\}^N$, which is a straightforward step (problem II.1 in ref. 17), and then constructing the tensor from the Waring decomposition (a symmetric form of the tensor rank decomposition) given by these vectors

$$\mathcal{T} = \sum_{r=1}^R \mathbf{u}^{(r)} \otimes \mathbf{u}^{(r)} \otimes \mathbf{u}^{(r)},$$

where R is the number of T gates, ' \otimes ' denotes the outer product and the additions are under modulo 2.

Crucially, any alternative Waring decomposition of \mathcal{T} can also be mapped to a quantum circuit, as each vector $\mathbf{u}^{(r)}$ corresponds one-to-one to a T gate (this is because its action can be interpreted as a single-qubit $\pi/4$ -phase rotation after a change of basis, implemented by a T gate and a set of Clifford gates, respectively). The resulting circuit is equivalent to the original one up to Clifford operations¹⁹, albeit with a different number of T gates—according to the number of factors R in the decomposition. Hence, we recast the problem of T-count optimization as finding low-rank decompositions of \mathcal{T} .

TensorGame

AlphaTensor-Quantum, built upon AlphaTensor²⁵, finds low-rank Waring decompositions of tensors using deep RL. AlphaTensor-Quantum recasts the problem of tensor decomposition into a single-player game, called TensorGame, in which at each step s the player observes the game state, consisting of a tensor $\mathcal{T}^{(s)}$ and all the previously played actions (initially, the tensor $\mathcal{T}^{(1)}$ is the signature tensor \mathcal{T} , and there are no previous actions). At step s , the player selects an action consisting of a vector (or factor) $\mathbf{u}^{(s)} \in \{0, 1\}^N$. Choosing an action affects the game state: the tensor $\mathcal{T}^{(s+1)} = \mathcal{T}^{(s)} - \mathbf{u}^{(s)} \otimes \mathbf{u}^{(s)} \otimes \mathbf{u}^{(s)}$, and $\mathbf{u}^{(s)}$ is included as the last played action. The goal of TensorGame is to reach the all-zero tensor in as few moves as possible. The game ends after S moves if that is the case and, by construction, the played

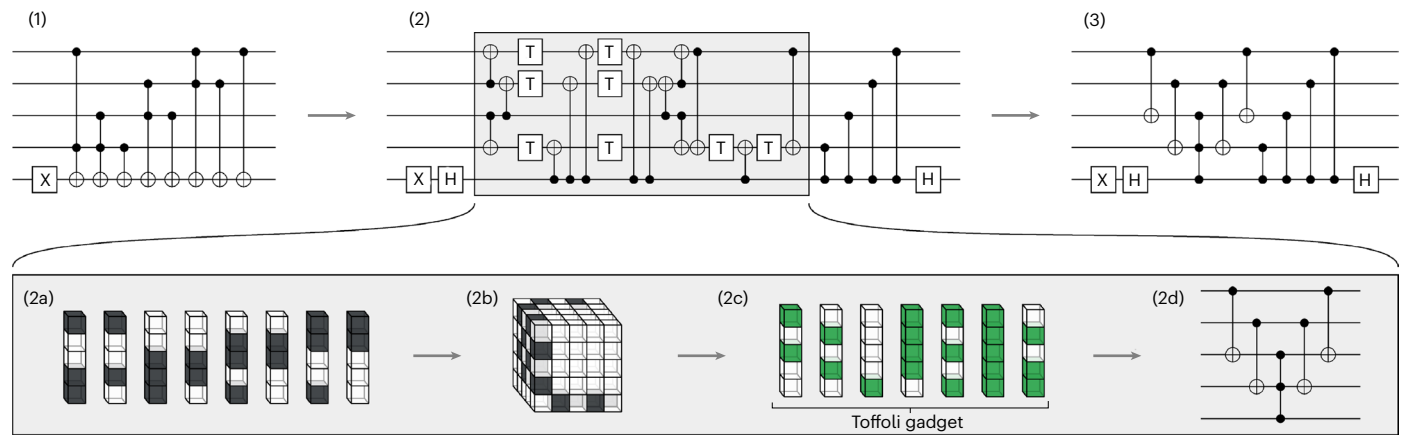


Fig. 3 | Results of AlphaTensor-Quantum on the Mod 5 circuit. (1) The original quantum circuit, which can be implemented with four Toffoli gates (with an equivalent cost of eight T gates). (2) The compiled circuit, which has been split into Clifford gates and a sub-circuit of just CNOT and T gates (in the highlighted box). (2a) The Waring decomposition corresponding to the highlighted CNOT+T circuit: each T gate corresponds precisely to one of the factors displayed as columns of blocks. (2b) The signature tensor formed from this decomposition. (2c) An alternative Waring decomposition of the tensor found by the RL agent.

Again, each factor corresponds one-to-one to a T gate; however, in this case, the seven factors are grouped into a single Toffoli gadget. (2d) The quantum circuit obtained from this new decomposition (implemented using a CCZ gate as the Toffoli and the CCZ gates are equivalent up to two Hadamard gates in the target qubit⁴³). (3) The optimized circuit after having applied AlphaTensor-Quantum, which can be implemented using Clifford gates and a single Toffoli gate (that is, its equivalent T-count is 2).

actions form a rank- S Waring decomposition of the signature tensor, as $\mathcal{T} = \sum_{s=1}^S \mathbf{u}^{(s)} \otimes \mathbf{u}^{(s)} \otimes \mathbf{u}^{(s)}$. (The game also ends after a user-specified maximum number of moves to avoid infinitely long games, in which case the game is declared ‘unsuccessful’ as it did not find a decomposition.) To encourage finding low-rank decompositions or, equivalently, circuits with lower T-count, games are penalized based on the number of moves taken to reach the all-zero tensor (with an additional penalty for unsuccessful games): every played move incurs a reward of -1 .

In AlphaTensor-Quantum, each action consists of a single factor $\mathbf{u}^{(s)}$, unlike standard AlphaTensor (where actions consist of factor triplets); this represents an important advantage because the action space reduces from 2^{3N} to 2^N , allowing AlphaTensor-Quantum to scale up to larger tensors (we consider sizes up to $N = 72$).

Gadgets

Unlike other T-count optimization approaches, AlphaTensor-Quantum can leverage gadgetization techniques to find more efficient constructions. In this context, a gadget is an alternative implementation of a quantum gate that reduces the number of T gates at the cost of introducing additional (ancilla) qubits and operations such as Clifford gates and measurement-based corrections. (These extra qubits do not represent a considerable limitation in practice, as they can be reused throughout the circuit.)

We consider two such gadgets. First, the Toffoli gate, which needs at least seven T gates¹⁶ to implement without ancillae but admits a gadget to implement it at a cost equivalent to that of two T gates^{12,42}. Second, the CS gate, whose direct implementation requires three T gates but whose gadgetized version has a cost equivalent to that of two T gates (Appendix A.3 in ref. 43).

Although the gadgetization techniques fall outside of the tensor decomposition framework, AlphaTensor-Quantum incorporates them into the reward signal of TensorGame. Every time that an action is played, we check whether it completes a Toffoli when combined with the six previous actions, a CS gate when combined with the two previous actions, or none of them. These checks are up to Clifford operations and they simply involve finding linear dependencies among the factors; see Methods for details and Fig. 2 for an illustration. If it completes a Toffoli, we assign a positive reward, so that the 7 actions jointly incur a reward of -2 (corresponding to 2 T gates) as opposed to the reward of -7 corresponding to the un-gadgetized implementation.

If it completes a CS, we adjust the reward so that the three actions jointly incur a reward of -2 .

Other than the reward signal, AlphaTensor-Quantum has no further information about the gadgets. It learns to recognize and exploit those patterns of actions from the data, and it finds good trade-offs between playing actions that are part of a gadget and actions that are not.

Results

We first demonstrate the effectiveness of AlphaTensor-Quantum on a benchmark of circuits widely used in the literature on T-count optimization, and then we show applications in different domains.

Benchmark circuits

We first consider a benchmark set of quantum circuits¹⁵ widely used in the literature^{17–19,24}, which we take from ref. 44. We apply two versions of AlphaTensor-Quantum: the full approach and an alternative agent that does not consider any gadgets.

Hadamard gadgetization. Some circuits contain Hadamard gates, which correspond to non-diagonal unitary matrices and are not in the CNOT+T gate set; thus they cannot be directly mapped to a signature tensor^{17,19,24}. To address this, we replace the Hadamard gates with other Clifford gates and measurement-based corrections, a process that does not alter the circuit behaviour but requires extra ancilla qubits (Supplementary Section C.1). We refer to this as Hadamard gadgetization¹⁷ to distinguish it from the gadgetization described in the previous section.

Circuit compilation. We apply a standard compilation technique^{17,24} with an improvement⁴¹ to obtain circuits containing only CNOT and T gates (see Supplementary Section C.1 for details and Fig. 3 for an example). We split circuits exceeding 60 qubits after Hadamard gadgetization into smaller sub-circuits of fewer than 60 qubits each, and apply AlphaTensor-Quantum on each sub-circuit independently (Supplementary Section C.2).

Results. In Table 1, we compare the results of the non-gadgetized version of AlphaTensor-Quantum against the best T-count reported in the literature^{15,17–20,23,24,45,46}, as the baselines do not consider

Table 1 | T-count achieved by different methods on a set of benchmark circuits

Circuit	Number of qubits		T-count without gadgets		T-count with gadgets	
	Original	Compiled	Baselines	AlphaTensor-Quantum	Original	AlphaTensor-Quantum
8-bit adder	24	*	129 (ref. 17)	139	114 (57Tof)	94 (33Tof + 28T)
Barenco Toff ₃	5	8	13 (refs. 23,24)	13 ^b	8 (4Tof)	4 (2Tof)
Barenco Toff ₄	7	14	24 (refs. 17,23)	23	16 (8Tof)	8 (4Tof)
Barenco Toff ₅	9	20	34 (refs. 17,23)	33	24 (12Tof)	12 (6Tof)
Barenco Toff ₁₀	19	50	84 (ref. 17)	83	64 (32Tof)	32 (16Tof)
CSLA-MUX ₃	15	21	40 (ref. 23)	39	20 (10Tof)	16 (8Tof)
CSUM-MUX ₉	30	42	72 (refs. 17,20)	71	56 (28Tof)	28 (14Tof)
GF(2 ²)-mult ^c	6	6	17 ^a	17 ^b	8 (4Tof)	6 (3Tof)
GF(2 ³)-mult ^c	9	9	31 ^a	29	18 (9Tof)	12 (6Tof)
GF(2 ⁴)-mult	12	12	47 (ref. 23)	39	32 (16Tof)	18 (9Tof)
GF(2 ⁵)-mult	15	15	84 (ref. 23)	59	50 (25Tof)	26 (13Tof)
GF(2 ⁶)-mult	18	18	118 (ref. 23)	77	72 (36Tof)	36 (18Tof)
GF(2 ⁷)-mult	21	21	167 (ref. 24)	104	98 (49Tof)	44 (22Tof)
GF(2 ⁸)-mult	24	24	214 (ref. 20)	123	128 (64Tof)	58 (29Tof)
GF(2 ⁹)-mult	27	27	295 (ref. 17)	161	162 (81Tof)	70 (35Tof)
GF(2 ¹⁰)-mult	30	30	350 (ref. 17)	196	200 (100Tof)	92 (46Tof)
Grover ₅	9	*	44 (ref. 17)	152	96 (48Tof)	66 (27Tof + 12T)
Hamming ₁₅ (high)	20	*	787 ^a (1,010 (ref. 17))	773	702 (351Tof)	440 (173Tof + 2CS + 90T)
Hamming ₁₅ (low)	17	42	75 (ref. 17)	73	46 (23Tof)	34 (17Tof)
Hamming ₁₅ (med)	17	*	156 ^a (162 (ref. 17))	156	164 (82Tof)	78 (35Tof + 8T)
HWB ₆	7	27	51 (ref. 17)	51	30 (15Tof)	20 (10Tof)
Mod-Adder ₁₀₂₄	28	*	798 ^a (978 (ref. 17))	762	570 (285Tof)	500 (141Tof + 15CS + 188T)
Mod-Mult ₅₅	9	11	17 (ref. 17)	17 ^b	14 (7Tof)	6 (3Tof)
Mod-Red ₂₁	11	28	55 (refs. 17,23)	51	34 (17Tof)	22 (11Tof)
Mod 5 ₄	5	5	7 (refs. 20,23,24)	7 ^b	8 (4Tof)	2 (1Tof)
QCLA-Adder ₁₀	36	*	116 (ref. 17)	135	68 (34Tof)	94 (28Tof + 5CS + 28T)
QCLA-Com ₇	24	42	59 (ref. 17)	59	58 (29Tof)	24 (12Tof)
QCLA-Mod ₇	26	*	165 (ref. 17)	199	118 (59Tof)	122 (43Tof + 36T)
QFT ₄	5	43	55 (ref. 17)	53	59 (2Tof + 55T)	44 (4Tof + 3CS + 30T)
RC-Adder ₆	14	24	37 (refs. 17,23)	37	22 (11Tof)	12 (6Tof)
NC Toff ₃	5	7	13 (refs. 17,23,24)	13 ^b	6 (3Tof)	4 (2Tof)
NC Toff ₄	7	11	19 (refs. 17,23,24)	19 ^b	10 (5Tof)	6 (3Tof)
NC Toff ₅	9	15	25 (ref. 17)	25	14 (7Tof)	8 (4Tof)
NC Toff ₁₀	19	35	55 (ref. 17)	55	34 (17Tof)	18 (9Tof)
VBE-Adder ₃	10	14	20 (refs. 17,23,24)	19 ^b	20 (10Tof)	6 (3Tof)

Numbers in bold indicate the best T-count for each case (with or without gadgets). Even without gadgetization, AlphaTensor-Quantum matches or outperforms the considered baselines for all circuits that have not been split into sub-circuits (split circuits are marked with a star). When considering gadgets, AlphaTensor-Quantum further reduces the T-count drastically and generally outperforms the original constructions with gadgets. The notation ‘aTof+bCS+cT’ indicates a circuit with a Toffoli gates, b CS gates and c T gates (when multiple solutions from AlphaTensor-Quantum achieve the same T-count but differ in the trade-off between gadgets and T gates, we report the result with the highest number of Toffoli gates). ^aBaseline results were obtained with the methods in Supplementary Section B and are better than the best-published T-count (provided in parentheses where applicable). ^bResults were proven to be optimal decompositions of the given tensor using the Z3 theorem prover. ^cThe GF(2²)-mult and GF(2³)-mult circuits are included for completeness (Supplementary Section D.1).

any gadgets either. For the circuits that did not require splitting, AlphaTensor-Quantum matches or outperforms all the existing T-count optimization methods. For all small circuits (at most 20 T gates), we confirmed with the Z3 theorem prover⁴⁷ that AlphaTensor-Quantum obtains the best possible T-count achievable with a tensor decomposition approach. For the circuits that exceed 60 qubits, AlphaTensor-Quantum outperforms the baselines for Hamming₁₅ (high) and Mod-Adder₁₀₂₄, but not for the others, probably due to the lack of optimality of the splitting technique (this is supported by the

fact that AlphaTensor-Quantum outperforms the baselines on individual sub-circuits; Supplementary Section D.2).

When considering gadgets, AlphaTensor-Quantum leverages the Toffoli and CS gates to further drastically reduce the T-count. When compared against the original construction of the circuits, consisting mostly of Toffoli gates, AlphaTensor-Quantum reduces the T-count in all except two cases (probably due to the splitting). When compared against the baselines, AlphaTensor-Quantum reduces the T-count for all circuits except Grover₅. Remarkably, for the GF(2^m)-mult circuits with

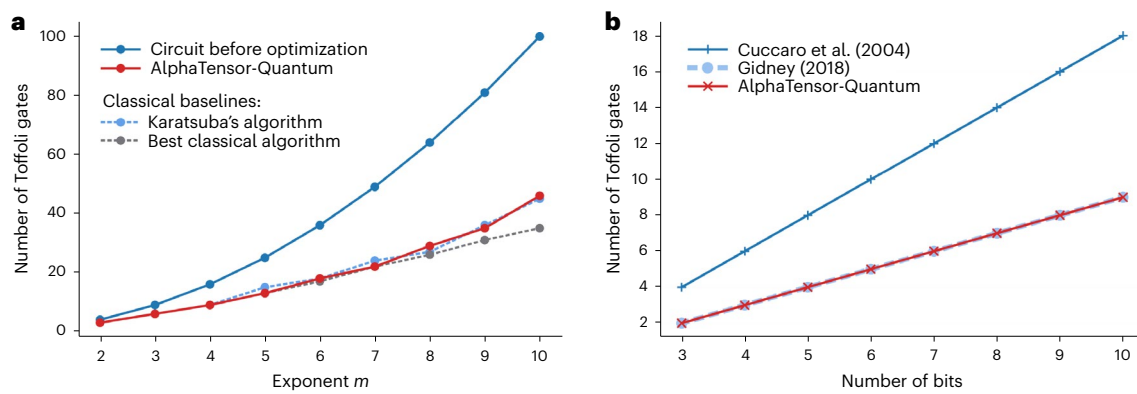


Fig. 4 | Number of Toffoli gates of the optimized circuits found by AlphaTensor-Quantum. **a**, For multiplication in finite fields of order 2^m , AlphaTensor-Quantum finds efficient circuits that substantially outperform the original construction, which scales like $\mathcal{O}(m^2)$. The number of Toffoli gates matches the best-known lower bound of the classical circuits⁵⁰ for some values of m , and scales as $\sim m^{\log_2(3)}$, showing that AlphaTensor-Quantum found an algorithm with the same complexity as Karatsuba's method³⁵, a classical algorithm for multiplication on finite fields for which a quantum version has not been reported in the literature. (The baselines are classical circuits, and

hence not directly comparable, as naive translations of classical to quantum circuits commonly introduce overheads. To compare, we assume the number of effective Toffoli gates is the number of 1-bit AND gates in the classical circuit.) **b**, For binary addition, AlphaTensor-Quantum halves the cost of the circuits from Cuccaro et al. (2004)⁵¹ matching the state-of-the-art circuits from Gidney (2018)³⁸. Remarkably, it does so automatically without any prior knowledge of the measurement-based uncomputation technique, which was crucial to their results.

the considered values of the parameter m , AlphaTensor-Quantum finds constructions with substantially smaller T-count than the baselines or the original constructions; we discuss this further below.

Applications

Multiplication in finite fields. Multiplication in finite fields is a crucial operation for cryptography, as many elliptic-curve cryptography implementations are built over finite fields, and there is a quantum algorithm that cracks elliptic-curve cryptography in polynomial time³⁴. Table 1 shows that AlphaTensor-Quantum excels in optimizing the primitives for multiplication in finite fields of order 2^m , that is, the circuits $\text{GF}(2^m)\text{-mult}$. In Fig. 4a, we show that for $m = \{2, 3, 4, 5, 7\}$, AlphaTensor-Quantum achieves the same Toffoli gate count as the best-known upper bound for the equivalent classical (non-reversible) circuits^{48,49}. For $m = \{2, 3, 4, 5\}$ this is also the best-known lower bound⁵⁰, which suggests that the resulting quantum circuits are likely optimal. While the circuits for these targets were constructed using the naive $\mathcal{O}(m^2)$ multiplication algorithm, the number of Toffoli gates of the optimized circuits follows $\sim m^{\log_2(3)}$ for the considered values of m . Thus, AlphaTensor-Quantum found a multiplication algorithm with the same complexity as Karatsuba's method³⁵, a classical algorithm for multiplication in finite fields. While it is conceivable to construct the optimized circuits by hand, no sub-quadratic constructions of quantum circuits for multiplication in finite fields have been discussed in the literature. AlphaTensor-Quantum finds them automatically, similar to Gidney's translation of Karatsuba's method for multiplication of integers³⁷, but without requiring ancilla qubits.

Binary addition. Binary addition is a relevant primitive because other common operations rely on it (such as multiplication, comparators or modular arithmetic). For example, controlled addition circuits are the dominant cost of Shor's algorithm. We focus on ripple-carry addition (instead of carry-save or block-carry-lookahead) because this requires fewer extra ancilla qubits from Hadamard gadgetization. In particular, we optimize the circuits of ref. 51, which have a Toffoli count scaling as $2n + \mathcal{O}(1)$ for n input bits, and the circuits of ref. 38, with a Toffoli count of $n + \mathcal{O}(1)$. As shown in Fig. 4b, AlphaTensor-Quantum halves the Toffoli count of the circuits in ref. 51, matching that of ref. 38.

Curcaco et al.'s circuits⁵¹ attained the lowest Toffoli T-count of any addition circuit for more than a decade until ref. 38 halved their cost by

introducing an approach called measurement-based uncomputation (which allows Toffoli gates acting on zero-initialized ancilla qubits to be uncomputed for free under certain conditions, by using measurements and classically controlled Clifford gates)—a technique that is now widely used for optimizing fault-tolerant quantum circuits.

Importantly, for both circuit constructions, AlphaTensor-Quantum had no knowledge of measurement-based uncomputation, and despite all Toffoli gates being compiled naively with seven T gates, it automatically finds optimized circuits only accessible via measurement-based uncomputation, due to its ability to leverage gadgets, and exploiting the fact that ancilla qubits from the Hadamard gadgetization allow the movement of Toffoli gates (see Fig. 5 for an example).

Hamming weight and phase gradients. Applying small angle rotations to qubits is a common operation in quantum algorithms, for example, in the quantum Fourier transform, data loading and quantum chemistry. In fault-tolerant quantum circuits, any rotation can be implemented using the Solovay–Kitaev algorithm with a sequence of Hadamard and T gates. However, if the same small angle is to be applied repeatedly (such as in quantum chemistry), it can be more efficient to use a technique called Hamming weight phasing.

We use AlphaTensor-Quantum to optimize the Hamming weight phasing circuits from ref. 38. While the original circuits use measurement-based uncomputation, we instead compile a version where every Toffoli gate is constructed explicitly, as before. Again, AlphaTensor-Quantum halves the cost with respect to the naive input circuits, thus matching the complexity of the circuits with measurement-based uncomputation tricks.

Unary iteration. We also apply AlphaTensor-Quantum to a family of circuits called unary iteration⁴⁰, which implements the quantum equivalent of a classical n -to- 2^n demultiplexer or decoder. Specifically, these circuits allow selecting which Pauli operator (from a set of operators) to apply to the data qubits, based on the value of some control qubits. We apply this construction to the Pauli operators $P_1 = XI \cdots I$, $P_2 = IX \cdots I$, ..., $P_m = II \cdots X$ on $m = 8, 16$ and 32 qubits, that is, for $n = 3, 4$ and 5 , where I is the identity matrix and X denotes the (Clifford) Pauli-X gate (Extended Data Fig. 3). These circuits could be used to implement the same construction for any set of $8, 16$ or 32 Pauli operators, without extra T-count, and because of their generality, they appear both in

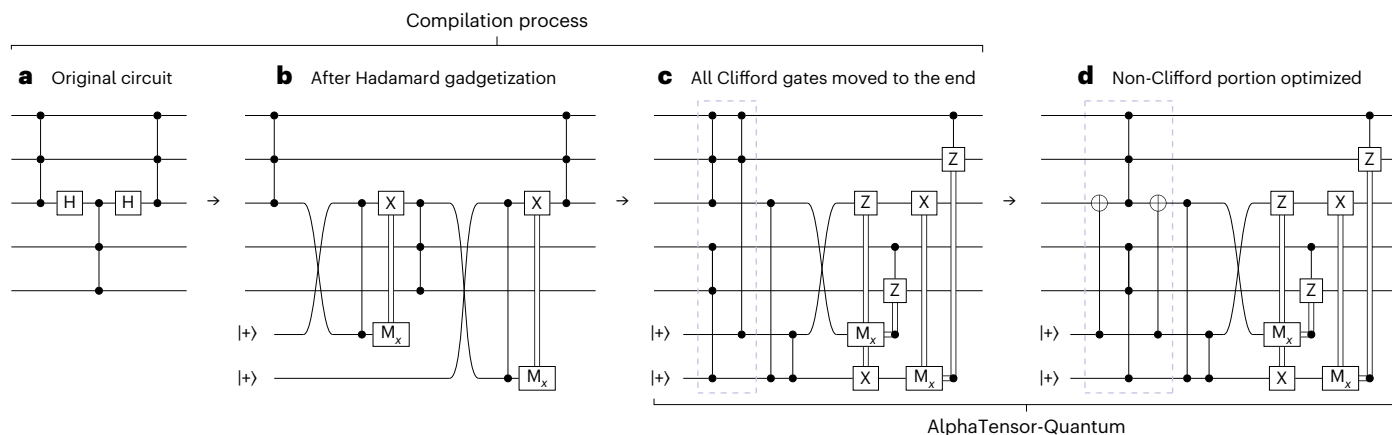


Fig. 5 | An example in which measurement-based uncomputation tricks can be recovered by AlphaTensor-Quantum. **a**, The original NC Toff₃ circuit. **b**, The Hadamard gadgetization process maps a portion of the circuit to three CCZ gates. **c**, After moving all Clifford gates to the end, two of the CCZ gates (in the highlighted box) share two inputs. **d**, AlphaTensor-Quantum optimizes this pair by expressing it as a single gadget, saving one Toffoli gate with respect to

the original construction. This kind of optimization happens often because CCZ gates can usually be moved freely along the circuit, as the corresponding unitary matrix will be diagonal after compilation. This is reminiscent of the temporary logical-AND construction from ref. 38, although some patterns with more than two CCZ gates can also be combined in this way.

oracle construction (demultiplexers are commonly used in classical logic) and in quantum chemistry (as in the linear combination of unitaries method discussed further below).

We implement the unary iteration circuits following ref. 40, except we compile the Toffoli gates unitarily with seven T gates instead of using measurement-based uncomputation. The largest circuit ($n = 5$) has 72 qubits after compilation. As before, AlphaTensor-Quantum finds its own version of measurement-based uncomputation to match the best-known T-count of these circuits.

We also apply a version of AlphaTensor-Quantum without gadgets. Remarkably, it finds optimized circuits that, for each value of n , have only three T gates more than the circuits from ref. 40 (where each Toffoli gate is implemented using four T gates), suggesting that the unary iteration circuits have an internal structure particularly amenable to this kind of optimization.

Quantum chemistry. Molecular simulation involves solving the Schrödinger equation, which requires a computational cost scaling exponentially with the number of particles in the system. This makes classical methods struggle to simulate strongly correlated molecules⁵². Quantum computers can theoretically handle these calculations more efficiently, with one compelling use case being the simulation of the FeMoco molecule, the active site in the nitrogenase enzyme responsible for nitrogen fixation. This use case has high potential industrial impact, but is challenging due to the highly complex electronic structure of FeMoco³⁹.

A method by ref. 53 simulates FeMoco using a fault-tolerant implementation of a quantum walk algorithm, where the Hamiltonian associated with the molecule is encoded into the quantum circuit with a technique known as linear combination of unitaries. This framework requires the implementation of two oracles called PREPARE and SELECT (each being an instance of the unary iteration operation discussed earlier). These are repeated many times, making up the dominant cost of the whole algorithm, and therefore optimizing these oracles can noticeably impact the total resource requirements (see Supplementary Section A.2 for more details on the algorithm).

We run AlphaTensor-Quantum on portions of the SELECT oracle construction in ref. 53 (which is itself based on ref. 54). Specifically, we set a range of parameters to keep the total qubit count below 60, and compile angle rotations of the state into the diagonal basis using the phase-gradient technique from ref. 38; see Extended Data Fig. 4 for details. As before, we remove all measurement-based uncomputation

tricks and compile the Toffoli gates unitarily with seven T gates. We find that, for all parameter values, AlphaTensor-Quantum matches the T-count of ref. 53, which relies heavily on measurement-based uncomputation.

Thus, AlphaTensor-Quantum automatically finds state-of-the-art constructions for multiple use cases. We expect that, as new improvements on quantum chemistry are made at a structural level, AlphaTensor-Quantum will be able to automatically find the best constructions (in terms of T-count), without additional human effort.

Discussion

We have demonstrated the effectiveness of AlphaTensor-Quantum at optimizing the T-count, showing that it substantially outperforms existing approaches, being able to automatically discover constructions as efficient as the best human-designed ones across various applications. For those applications, the fact that very different techniques achieve the same T-count might hint at the optimality of the discovered constructions. In fact, we proved that some decompositions are optimal in terms of the number of factors.

Compared with previous studies that use RL to optimize quantum circuits^{26–28} or variational quantum algorithms^{29–33}, AlphaTensor-Quantum is the first model trained on circuits with a large number of qubits to cover a wide range of applications, and the first one considering fault-tolerant compilation. Compared with existing T-count optimization approaches, AlphaTensor-Quantum substantially outperforms them—at the cost of increased computational complexity (for example, it took around 8 hours to optimize the circuits of ref. 51) and loss of optimality due to extra compilation steps (for example, if the circuits need to be split into parts). Future research avenues to accelerate the algorithm include improving the neural network architecture, handling Hadamard gadgetization differently (to keep the number of qubits small enough), or applying RL to alternative circuit representations that can optimize Hadamard gates natively. However, its comparably increased computational complexity is outweighed by its results, especially considering the cost of manually designing optimized constructions. Moreover, it is a one-off cost, as once discovered, the optimized circuits can be reused.

Unlike existing approaches, AlphaTensor-Quantum leverages and exploits gadgets. These constructions are an active area of research^{42,43,55,56}, and as fundamentally new gadgets are discovered, they can be readily incorporated into the RL environment, possibly allowing AlphaTensor-Quantum to discover even more efficient constructions

for established quantum circuits. Similarly, AlphaTensor-Quantum can be extended to optimize metrics other than the T-count, for example, T-depth⁵⁷ or weighted combinations of different gates¹⁸, by changing the reward signal accordingly.

We envision that AlphaTensor-Quantum will play a pivotal role as quantum chemistry and related fields advance, and new improvements at a structural level emerge. Similarly to how Gidney³⁸ optimized the addition circuits of Cuccaro et al.⁵¹ over a decade later, AlphaTensor-Quantum finds and exploits non-trivial patterns, eliminating the need for manual construction design and saving research costs.

Methods

The signature tensor

The tensor representation of a circuit relies on the concept of phase polynomials. Consider a circuit with CNOT and T gates alone. Its corresponding unitary matrix U performs the operation $U|x_{1:N}\rangle = e^{i\phi(x_{1:N})}|Ax_{1:N}\rangle$ where $\phi(x_{1:N})$ is the phase polynomial and A is an invertible matrix that can be implemented with Clifford gates only. For example, applying a T gate to the second qubit after a CNOT gives $(I \otimes T)\text{CNOT}|x, y\rangle = e^{i\frac{\pi}{4}(x \oplus y)}|x, x \oplus y\rangle$ that is, $\phi(x, y) = \frac{\pi}{4}(x \oplus y) = \frac{\pi}{4}(x + y - 2xy)$. After cancelling out the terms leading to multiples of 2π , the phase polynomial takes a multilinear form, that is, $\phi(x_{1:N}) = \frac{\pi}{4}(\sum_i a_i x_i + 2\sum_{i < j} b_{ij} x_i x_j + 4\sum_{i < j < k} c_{ijk} x_i x_j x_k)$, where $a_i \in \{0, \dots, 7\}$, $b_{ij} \in \{0, \dots, 3\}$ and $c_{ijk} \in \{0, 1\}$. This maps directly to a circuit of T, CS and CCZ gates: each $a_i x_i$ term is implemented by a_i T gates acting on the i th qubit, each $b_{ij} x_i x_j$ term corresponds to b_{ij} CS gates on qubits i and j , and each non-zero $c_{ijk} x_i x_j x_k$ term is a CCZ gate on qubits i, j and k . Each of these terms corresponds to a non-Clifford operation only if the corresponding constant (a_i , b_{ij} or c_{ijk}) is odd. Thus, two CNOT+T circuits implement the same unitary up to Clifford gates if and only if they have the same phase polynomial after taking the modulo 2 of their coefficients. This information about the polynomial can then be captured in a symmetric three-dimensional binary signature tensor \mathcal{T} of size $N \times N \times N$ (refs. 17,19). See Supplementary Section A.1 for more details.

We remark that the signature tensor captures information about the non-Clifford components of the circuit only. Thus, two circuits with the same signature tensor may implement a different unitary matrix—but it is always possible to implement one of the circuits by adding Clifford gates to the output of the other circuit. As in this work we consider the T gate as the only metric of complexity, and Clifford gates do not incur any cost, we consider two circuits with the same signature tensor to be equivalent for the purposes of T-count optimization.

System description

AlphaTensor-Quantum builds on AlphaTensor, which is in turn based on AlphaZero⁵⁸. It is implemented over a distributed computing architecture, composed of one learner and multiple actors that communicate asynchronously. The learner is responsible for hosting a neural network and updating its parameters, while the main role of the actors is to play games to generate data for the learner. When the learner updates the network parameters by gradient descent steps, it sends them to the actors. The actors employ Monte Carlo tree search (MCTS), using queries to the network to guide their policy, and play games whose actions tend to improve over the policy dictated by the neural network. Played games are sent to the learner and stored in a buffer, and they will serve as future data points to train the network parameters.

Neural network. The neural network at the core of AlphaTensor-Quantum takes as input the game state and outputs a policy, that is, a probability distribution over the next action to play, and a probability distribution over the estimated return (that is, the sum of rewards from the current state until the end of the game) (Extended Data Fig. 5). The neural network has a key component, the torso, where a set of attention operations⁵⁹ are applied. In AlphaTensor-Quantum, the torso interleaves two types of layer: axial attention⁶⁰ and symmetrization layers

(Extended Data Fig. 5). Symmetrization layers are inspired by the empirical observation that, for a symmetric input $X \in \mathbb{R}^{N \times N}$ (with $X = X^T$), preserving the symmetry of the output in-between axial attention layers substantially improves performance. Therefore, after each axial attention layer, we add a symmetrization operation, defined as $X \leftarrow \beta \odot X + (1 - \beta) \odot X^T$, where ' \odot ' denotes the element-wise product. For $\beta = 1/2$, this makes the output X symmetric. In practice, we found that letting β be a learnable $N \times N$ matrix improves performance even further, even though X is no longer symmetric, due to the ability of the network to exchange information between rows and columns after each axial attention layer. (When the input is a tensor, $X \in \mathbb{R}^{N \times N \times C}$, we apply the symmetrization operations independently across the dimensions C , using the same matrix β .) We refer to this architecture as symmetrized axial attention. See Supplementary Section C.4 for more details of the full neural network architecture.

Symmetrized axial attention performs better than the neural network of standard AlphaTensor (which requires attention operations across every pair of modes of the tensor)²⁵. It is also one of the key ingredients that allow AlphaTensor-Quantum to scale up to larger tensors: for $N = 30$ and the same number of layers, symmetrized axial attention runs about 4 times faster and reduces the memory consumption by a factor of 3. (See also Supplementary Section D.3 for further ablations.) We believe symmetrized axial attention may be useful for other applications beyond AlphaTensor-Quantum.

Gadgetization. The Toffoli gate admits a gadget to implement it using four T gates⁴² instead of the seven T gates of the ancilla-free implementation. When considering a state-of-the-art magic-state factory⁴², which converts a CCZ magic state into two T states, we can consider that the cost of the Toffoli gadget is equivalent to that of two T gates. Similarly, the CS gate can also be built using Clifford operations and a single (CCZ) magic state, so that its cost is equivalent to that of two T gates when considering a state-of-the-art magic-state factory⁴².

AlphaTensor-Quantum incorporates gadgetization into TensorGame as follows. Every time that an action is played, we check whether it completes a Toffoli when combined with the six previous actions, a CS gate when combined with the two previous actions, or none of them. If it completes a Toffoli, we assign a positive reward of +4, so that the 7 actions jointly incur a reward of −2 (corresponding to 2 T gates) as opposed to the reward of −7 corresponding to the un-gadgetized implementation. If it completes a CS, we assign a reward of 0, so that the 3 actions jointly incur a reward of −2. See Fig. 2 for an illustration.

We can check whether a gadget was completed, up to Clifford operations, by simply finding linear dependencies between $\mathbf{u}^{(s)}$ and the previous actions (see Supplementary Section C.3 for a justification):

- If $s \geq 7$, we check whether the action $\mathbf{u}^{(s)}$ completes a Toffoli gadget. Let $\mathbf{a} \triangleq \mathbf{u}^{(s-6)}$, $\mathbf{b} \triangleq \mathbf{u}^{(s-5)}$ and $\mathbf{c} \triangleq \mathbf{u}^{(s-4)}$. If the vectors \mathbf{a} , \mathbf{b} and \mathbf{c} are linearly independent and all the following relationships hold, then we declare that $\mathbf{u}^{(s)}$ completes a Toffoli gadget:

$$\begin{aligned}\mathbf{u}^{(s-3)} &= \mathbf{a} + \mathbf{b}, & \mathbf{u}^{(s-2)} &= \mathbf{a} + \mathbf{c}, \\ \mathbf{u}^{(s-1)} &= \mathbf{a} + \mathbf{b} + \mathbf{c}, & \text{and} \quad \mathbf{u}^{(s)} &= \mathbf{b} + \mathbf{c}.\end{aligned}$$

- If $s \geq 3$, we check whether the action $\mathbf{u}^{(s)}$ completes a CS gadget. Let $\mathbf{a} \triangleq \mathbf{u}^{(s-2)}$ and $\mathbf{b} \triangleq \mathbf{u}^{(s-1)}$. If the vectors \mathbf{a} and \mathbf{b} are linearly independent and $\mathbf{u}^{(s)} = \mathbf{a} + \mathbf{b}$, then $\mathbf{u}^{(s)}$ completes a CS gadget.

We refer to the relationships above as gadgetization patterns. Each factor (action) may belong to at most one gadget; thus, if the action at step s completes a gadget of any type, we only check for a new CS gadget from step $s + 3$ onwards, and for a new Toffoli gadget from step $s + 7$ onwards.

Synthetic demonstrations. Like its predecessor, AlphaTensor-Quantum uses a dataset of randomly generated tensor/factorization

pairs to train the neural network. Specifically, the network is trained on a mixture of a supervised loss (which encourages it to imitate the synthetic demonstrations) and a standard RL loss (which encourages it to learn to decompose the target quantum signature tensor). For each experiment, we generate 5 million synthetic demonstrations. To generate each demonstration, we first randomly sample the number of factors R from a uniform distribution in [1, 125] and then sample R binary factors $\mathbf{u}^{(r)}$, such that each element has a probability of 0.75 of being set to 0.

After having generated the factors, we randomly overwrite some of them to incorporate the patterns of the Toffoli and CS gadgets, to help the network learn and identify those patterns. For each demonstration, we include at least one gadget with probability 0.9. The number of gadgets is uniformly sampled in [1, 15] (the value is also truncated when R is not large enough), and for each gadget we sample the starting index r and the type of gadget (Toffoli with probability 0.6 and CS with probability 0.4), overwriting the necessary factors according to the gadget's pattern. We ensure that gadgets do not overlap throughout the demonstration.

Change of basis. To increase the diversity of the played games, the actors apply a random change of basis to the target tensor \mathcal{T} at the beginning of each game. From the quantum computation point of view, a change of basis can be implemented by a Clifford-only circuit, and therefore this procedure does not affect the resulting number of T gates.

A change of basis is represented by a matrix $M \in \{0, 1\}^{N \times N}$ such that $\det(M) = 1$ under modulo 2. We randomly generate 50,000 such basis changes and use them throughout the experiment, that is, the actor first randomly picks a matrix M and then applies the following transformation to the target tensor \mathcal{T} :

$$\tilde{\mathcal{T}}_{ijk} = \sum_{a=1}^N \sum_{b=1}^N \sum_{c=1}^N M_{ia} M_{jb} M_{kc} \mathcal{T}_{abc}.$$

The actor then plays the game in that basis (that is, it aims at finding a decomposition of $\tilde{\mathcal{T}}$). After the game has finished, we can map the played actions back to the standard basis by applying the inverse change of basis.

The diversity injected by the change of basis facilitates the agent's task because it suffices to find a low-rank decomposition in any basis. Note that gadgets are not affected by the change of basis, as their patterns correspond to linear relationships that are preserved after linear transformations.

Data augmentation. For each game played by the actors, we obtain a new game by swapping two of the actions in it. Specifically, we swap the last action that is not part of a gadget with a randomly chosen action that is also not part of a gadget. Mathematically, swapping actions is a valid operation due to commutativity. From the quantum computation point of view, it is a valid operation because the considered unitary matrices are diagonal.

Besides the action permutation, we also permute the inputs at acting time every time we query the neural network. In particular, we apply a random permutation to the input tensor and past actions that represent the current state, and then invert this permutation at the output of the network's policy head. Similarly, at training time, we apply a random permutation to both the input and the policy targets, and train the neural network on this transformed version of the data. In practice, we sample 100 permutations at the beginning of an experiment and use them thereafter.

Sample-based MCTS. In AlphaTensor-Quantum, the action space is huge, and therefore it is not possible to enumerate all the possible actions from a given state. Instead, AlphaTensor-Quantum relies on sample-based MCTS, similarly to sampled AlphaZero⁶¹.

Before committing to an action, the agent uses a search tree to explore multiple actions to play. Each node in the tree represents a state, and each edge represents an action. For each state–action pair (s, a) , we store a set of statistics: the visit count $N(s, a)$, the action value $Q(s, a)$ and the empirical policy probability $\hat{\pi}(s, a)$. The search consists of multiple simulations; each simulation traverses the tree from the root state s_0 until a leaf state s_L is reached, by recursively selecting in each state s an action a that has high empirical policy probability and high value but has not been frequently explored, that is, we choose a according to

$$\arg \max_a Q(s, a) + c(s) \hat{\pi}(s, a) \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)},$$

where $c(s)$ is an exploration factor. The MCTS simulation keeps choosing actions until a leaf state s_L is reached; when that happens, the network is queried on s_L —obtaining the policy probability $\pi(s_L, \cdot)$ and the return probability $z(s_L)$. We sample K actions a_k from the policy to initialize $\hat{\pi}(s_L, a) = \frac{1}{K} \sum_k \delta_{a, a_k}$, and initialize $Q(s_L, a)$ from $z(s_L)$ using a risk-seeking value²⁵. After the node s_L has been expanded, we update the visit counts and values on the simulated trajectory in a backwards pass⁶¹. We simulate 800 trajectories using MCTS; after that, we use the visit counts of the actions at the root of the search tree to form a sample-based improved policy, following ref. 25, and commit to an action sampled from that improved policy.

Training regime. We train AlphaTensor-Quantum separately for each considered application (that is, one agent for the benchmark circuits, another one for multiplication in finite fields, another one for all the circuits related to binary addition and so on). This approach allows the actors to repeatedly play enough games starting from the tensors of interest (becoming better and better at decomposing those specific targets), while at the same time being able to identify and exploit possible decomposition patterns shared across the circuits within the same application. We find that the experiment variance is very low: a single experiment is enough for AlphaTensor-Quantum to find the best constructions for each application.

We train AlphaTensor-Quantum on a tensor processing unit (TPU) with 256 cores, with a total batch size of 2,048, training for about 1 million iterations (the number of training iterations varies depending on the application, ranging between 80,000 and 3 million iterations). We generally use 3,600 actors per experiment, each running on a standalone TPU (experiments on the benchmark circuits were run with twice as many actors).

The computational cost of running an experiment depends on multiple factors. The main ones are: the number of qubits N , the episode length (that is, the maximum allowed number of steps in TensorGame), the circuit complexity (that is, how many steps it takes to optimally decompose the tensor) and the architecture parameters (for example, the number of layers of the torso). Theoretically, the cost increases roughly cubically with N , linearly with either the episode length (at the beginning of the training algorithm) or the circuit complexity (after a number of training iterations), and linearly with the number of layers of the torso. As an example, the training algorithm for the most costly experiment runs at about 2 steps per second (corresponding to the experiment on unary iteration, with $N = 72$, episode length of 250 and 4 layers).

Other versions of AlphaTensor-Quantum. We found empirically that the $\text{GF}(2^m)$ -mult circuits were particularly challenging to optimize, and they required additional training iterations. To reduce the size of the search space and accelerate the training procedure, we developed a variant of AlphaTensor-Quantum that favours Toffoli gates. Specifically, every time that three consecutive and linearly independent actions (factors) are played, the environment automatically applies the remaining four

actions that would complete the Toffoli gadget. We applied this version of AlphaTensor-Quantum to optimize these targets only.

Verification

To verify that the results of AlphaTensor-Quantum are correct, we used the ‘feynver’ tool developed by ref. 44 after each step of the compilation process discussed in Supplementary Section C.1. It uses the sum-over-paths formalism⁶² to produce a proof of equality and can be applied to any circuit. However, it is a sound but not complete method, so it may fail both to prove and to disprove equality for some pairs of circuits. As the tool runs slowly for larger circuits, we ran it wherever practical (limiting each run to several hours). Each circuit was verified assuming that any intermediate measurements introduced by Hadamard gadgetization always return zero, because our software pipeline does not generate the classically controlled correction circuits (these can be easily constructed but are irrelevant to AlphaTensor-Quantum).

This allowed us to verify inputs up to the point where a tensor is obtained. To verify that the decompositions constructed by AlphaTensor-Quantum were correct, we computed the corresponding tensor and checked that it was equal to the original input tensor. While we did not use the optimized decompositions to produce circuits, they could be verified in the same way as discussed above.

Data availability

The experiments carried out in this paper do not require any data corpus other than the publicly available quantum circuits, taken from refs. 38,40,44,51,53. Experiments with AlphaTensor-Quantum also require synthetic demonstrations, which can be procedurally generated as detailed in Methods. The Waring decompositions discovered by AlphaTensor-Quantum are available for download at https://github.com/google-deepmind/alphatensor_quantum and <https://doi.org/10.5281/zenodo.14679491> (ref. 63).

Code availability

A full description of AlphaTensor, the algorithm that this work builds on, is available in ref. 25. The details of the neural network architecture that we use are described in Methods and Supplementary Section C.4. The procedure to compile the quantum circuits is fully described in Supplementary Section C.1. We provide a repository at https://github.com/google-deepmind/alphatensor_quantum and <https://doi.org/10.5281/zenodo.14679491> (ref. 63) containing the following Python code. (1) The code for the innovations of AlphaTensor-Quantum over AlphaTensor, that is, the environment dynamics and gadgetization procedure, the neural network, the algorithm for generating synthetic demonstrations, as well as the generation of change of basis. (2) A runnable demo that shows how to connect the innovative components of AlphaTensor-Quantum to a third-party codebase: the library MCTX, which implements Monte Carlo tree search in Jax⁶⁴. The demo features a simplified version of AlphaTensor that runs on a single machine. In an additional repository at <https://github.com/tlaakkonen/circuit-to-tensor> and <https://doi.org/10.5281/zenodo.14680969> (ref. 65), we provide the following. (1) The code for the pipeline of Fig. 1: (i) transforming an input quantum circuit into a tensor, and (ii) transforming the output factorizations of AlphaTensor-Quantum back into a quantum circuit. (2) The code for verifying the correctness of our results, which would let users verify the correctness of transforming quantum circuits into tensors for all of the benchmark targets.

References

- Feynman, R. P. Simulating physics with computers. *Int. J. Theor. Phys.* **21**, 467–488 (1982).
- Manin, Y. I. *Computable and Noncomputable* (Sovetskoe Radio, 1980).
- Shor, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* **26**, 1484–1509 (1997).
- Blunt, N. S. et al. Perspective on the current state-of-the-art of quantum computing for drug discovery applications. *J. Chem. Theory Comput.* **18**, 7001–7023 (2022).
- Dalzell, A. M. et al. *Quantum Algorithms: a Survey of Applications and End-to-End Complexities*. (Cambridge Univ. Press, in the press).
- Gottesman, D. *The Heisenberg Representation of Quantum Computers* (Los Alamos National Laboratory, 1998); <https://arxiv.org/abs/quant-ph/9807006>
- Bravyi, S. & Kitaev, A. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Phys. Rev. A* **71**, 022316 (2005).
- Campbell, E. T., Terhal, B. M. & Vuillot, C. Roads towards fault-tolerant universal quantum computation. *Nature* **549**, 172–179 (2017).
- Calderbank, A. R. & Shor, P. W. Good quantum error-correcting codes exist. *Phys. Rev. A* **54**, 1098–1105 (1996).
- Aharonov, D. & Ben-Or, M. Fault tolerant quantum computation with constant error. In *Proc. 29th Annual ACM Symposium on Theory of Computing* 176–188 (ACM, 1997).
- Eastin, B. & Knill, E. Restrictions on transversal encoded quantum gate sets. *Phys. Rev. Lett.* <https://doi.org/10.1103/physrevlett.102.110502> (2009).
- Gidney, C. & Fowler, A. G. Efficient magic state factories with a catalyzed $|CCZ\rangle$ to $|CCZ\rangle$ transformation. *Quantum* **3**, 135 (2019).
- Gidney, C., Shutty, N. & Jones, C. Magic state cultivation: growing T states as cheap as CNOT gates. Preprint at <https://arxiv.org/abs/2409.17595> (2024).
- van de Wetering, J. & Amy, M. Optimising quantum circuits is generally hard. Preprint at <https://arxiv.org/abs/2310.05958> (2023).
- Amy, M., Maslov, D. & Mosca, M. Polynomial-time T-depth optimization of Clifford+T circuits via matroid partitioning. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* <https://doi.org/10.1109/TCAD.2014.2341953> (2014).
- Gosset, D., Kliuchnikov, V., Mosca, M. & Russo, V. An algorithm for the T-count. *Quantum Inf. Comput.* **14**, 1261–1276 (2014).
- Heyfron, L. E. & Campbell, E. T. An efficient quantum compiler that reduces T count. *Quantum Sci. Technol.* **4**, 015004 (2018).
- Nam, Y., Ross, N. J., Su, Y., Childs, A. M. & Maslov, D. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Inf.* **4**, 23 (2018).
- Amy, M. & Mosca, M. T-count optimization and Reed-Muller codes. *IEEE Trans. Inf. Theory* **65**, 4771–4784 (2019).
- Kissinger, A. & van de Wetering, J. Reducing the number of non-Clifford gates in quantum circuits. *Phys. Rev. A* <https://doi.org/10.1103/PhysRevA.102.022406> (2020).
- Gheorghiu, V., Mosca, M. & Mukhopadhyay, P. T-count and T-depth of any multi-qubit unitary. *npj Quantum Inf.* **8**, 141 (2022).
- Kissinger, A. & van de Wetering, J. Simulating quantum circuits with ZX-calculus reduced stabiliser decompositions. *Quantum Sci. Technol.* **7**, 044001 (2022).
- de Beaudrap, N., Bian, X. & Wang, Q. Techniques to reduce $\pi/4$ -parity-phase circuits, motivated by the ZX calculus. In *16th International Conference on Quantum Physics and Logic* 131–149 (EPTCS, 2020).
- de Beaudrap, N., Bian, X. & Wang, Q. Fast and effective techniques for T-count reduction via spider nest identities. In *Conference on the Theory of Quantum Computation, Communication and Cryptography* (ed. Flammia, S. T.) 11:1–11:23 (Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020).
- Fawzi, A. et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature* **610**, 47–53 (2022).

26. Moro, L., Paris, M. G. A., Restelli, M. & Prati, E. Quantum compiling by deep reinforcement learning. *Commun. Phys.* **4**, 178 (2021).
27. Fösel, T., Niu, M. Y., Marquardt, F. & Li, L. Quantum circuit optimization with deep reinforcement learning. Preprint at <https://arxiv.org/abs/2103.07585> (2021).
28. Quetschlich, N., Burgholzer, L. & Wille, R. Compiler optimization for quantum computing using reinforcement learning. In *2023 60th ACM/IEEE Design Automation Conference* 1–6 (IEEE, 2023).
29. Ostaszewski, M., Trenkwalder, L. M., Masarczyk, W., Scerri, E. & Dunjko, V. Reinforcement learning for optimization of variational quantum circuit architectures. *Adv. Neural Inf. Process. Syst.* **34**, 18182–18194 (2021).
30. Kuo, E.-J., Fang, Y.-L. L. & Chen, S. Y.-C. Quantum architecture search via deep reinforcement learning. Preprint at <https://arxiv.org/abs/2104.07715> (2021).
31. Yao, J., Li, H., Bukov, M., Lin, L. & Ying, L. Monte Carlo tree search based hybrid optimization of variational quantum circuits. In *Proc. Mathematical and Scientific Machine Learning* (eds Dong, B. et al.) 49–64 (PMLR, 2022).
32. Zhu, X. & Hou, X. Quantum architecture search via truly proximal policy optimization. *Sci. Rep.* **13**, 5157 (2023).
33. Rosenhahn, B. & Osborne, T. J. Monte Carlo graph search for quantum circuit optimization. *Phys. Rev. A* **108**, 062615 (2023).
34. Cheung, D., Maslov, D., Mathew, J. & Pradhan, D. K. On the design and optimization of a quantum polynomial-time attack on elliptic curve cryptography. In *Theory of Quantum Computation, Communication, and Cryptography* (eds Kawano, Y. & Mosca, M.) 96–104 (Springer, 2008).
35. Karatsuba, A. & Ofman, Y. Multiplication of many-digital numbers by automatic computers. *Proc. USSR Acad. Sci.* **145**, 293–294 (1962).
36. Bennett, C. H. Time/space trade-offs for reversible computation. *SIAM J. Comput.* **18**, 766–776 (1989).
37. Gidney, C. Asymptotically efficient quantum Karatsuba multiplication. (2019).
38. Gidney, C. Halving the cost of quantum addition. *Quantum* **2**, 74 (2018).
39. Reiher, M., Wiebe, N., Svore, K. M., Wecker, D. & Troyer, M. Elucidating reaction mechanisms on quantum computers. *Proc. Natl Acad. Sci. USA* **114**, 7555–7560 (2017).
40. Babbush, R. et al. Encoding electronic spectra in quantum circuits with linear T complexity. *Phys. Rev. X* **8**, 041015 (2018).
41. Vandaele, V., Martiel, S., Perdrix, S. & Vuillot, C. Optimal Hadamard gate count for Clifford + T synthesis of Pauli rotations sequences. *ACM Trans. Quantum Comput.* <https://doi.org/10.1145/3639062> (2024).
42. Jones, C. Low-overhead constructions for the fault-tolerant Toffoli gate. *Phys. Rev. A* **87**, 022328 (2013).
43. Beverland, M., Campbell, E., Howard, M. & Kliuchnikov, V. Lower bounds on the non-Clifford resources for quantum computations. *Quantum Sci. Technol.* **5**, 035009 (2020).
44. Amy, M. Feynman. Quantum circuit analysis toolkit. *GitHub* <https://github.com/meamy/feynman> (2016).
45. Zhang, F. & Chen, J. Optimizing T gates in Clifford + T circuit as $\pi/4$ rotations around Paulis. Preprint at <https://arxiv.org/abs/1903.12456> (2019).
46. Munson, A., Coecke, B. & Wang, Q. AND-gates in ZX-calculus: spider nest identities and QBC-completeness. In *Proc. 17th International Conference on Quantum Physics and Logic* (eds Valiron, B. et al.) 230–255 (EPTCS, 2020).
47. de Moura, L. & Björner, N. in *Tools and Algorithms for the Construction and Analysis of Systems* (eds Ramakrishnan, C. R. & Rehof, J.) 337–340 (Springer, 2008).
48. Montgomery, P. L. Five, six, and seven-term Karatsuba-like formulae. *IEEE Trans. Comput.* **54**, 362–369 (2005).
49. Fan, H. & Hasan, M. A. Comments on ‘five, six, and seven-term Karatsuba-like formulae’. *IEEE Trans. Comput.* **56**, 716–717 (2007).
50. Barbulescu, R., Detrey, J., Estivals, N. & Zimmermann, P. in *Arithmetic of Finite Fields* (eds Özbudak, F. & Rodríguez-Henríquez, F.) 168–186 (Springer, 2012).
51. Cuccaro, S. A., Draper, T. G., Kutin, S. A. & Moulton, D. P. A new quantum ripple-carry addition circuit. Preprint at <https://arxiv.org/abs/quant-ph/0410184> (2004).
52. Orús, R. Tensor networks for complex quantum systems. *Nat. Rev. Phys.* **1**, 538–550 (2019).
53. Lee, J. et al. Even more efficient quantum computations of chemistry through tensor hypercontraction. *PRX Quantum* **2**, 030305 (2021).
54. von Burg, V. et al. Quantum computing enhanced computational catalysis. *Phys. Rev. Res.* **3**, 033055 (2021).
55. Gidney, C. & Jones, C. A CCCZ gate performed with 6 T gates. Preprint at <https://arxiv.org/abs/2106.11513> (2021).
56. Amy, M. & Ross, N. J. Phase-state duality in reversible circuit design. *Phys. Rev. A* **104**, 052602 (2021).
57. Amy, M., Maslov, D., Mosca, M. & Roetteler, M. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **32**, 818–830 (2013).
58. Silver, D. et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* **362**, 1140–1144 (2018).
59. Vaswani, A. et al. Attention is all you need. In *Advances in Neural Information Processing Systems* (eds Guyon, U. et al.) (Curran Associates, 2017).
60. Ho, J., Kalchbrenner, N., Weissenborn, D. & Salimans, T. Axial attention in multidimensional transformers. Preprint at <https://arxiv.org/abs/1912.12180> (2019).
61. Hubert, T. et al. Learning and planning in complex action spaces. In *Proc. 38th International Conference on Machine Learning* 4476–4486 (PMLR, 2021).
62. Dawson, C. M. et al. Quantum computing and polynomial equations over the finite field Z_2 . *Quantum Inf. Comput.* **5**, 102–112 (2005).
63. Ruiz, F. J. R. AlphaTensor-Quantum GitHub repository. *Zenodo* <https://doi.org/10.5281/zenodo.14679491> (2025).
64. DeepMind et al. The DeepMind JAX ecosystem. *GitHub* <https://github.com/google-deepmind/mctx> (2020).
65. Laakkonen, T. Circuit-to-tensor GitHub repository. *Zenodo* <https://doi.org/10.5281/zenodo.14680969> (2025).

Acknowledgements

We thank S. Clark, A. L. Gaunt, A. Krajenbrink, L. Mondada and R. Yeung for their feedback on the paper; and R. Babbush, S. Boixo, C. Gidney, M. Harrigan, T. Khattar and N. Rubin for insightful discussions and ideas.

Author contributions

T.L. and J.v.d.W. conceived the project, which was kicked off with help from A.F., F.J.R.R., J.B., K.M. and P.K. F.J.R.R. and T.L. led the project. F.J.R.R. implemented the first working prototype of AlphaTensor-Quantum, developed the project codebase, ran the AlphaTensor-Quantum experiments and innovated on the neural network architecture, with support from A.F., A.N., B.R.-P., F.J.H.H., J.B., M. Balog and M. Barekatin. F.J.R.R., T.L. and J.B. came up with and designed the gadgetization techniques for AlphaTensor-Quantum, with the help of J.v.d.W. T.L. and J.v.d.W. analysed the results of AlphaTensor-Quantum. T.L. developed and ran the heuristics for circuit compilation, obtained the source circuits and ran the baseline

methods, with support from N.F. F.J.R.R., T.L., N.F., J.v.d.W and J.B. wrote the paper, with feedback from A.F., F.J.H.H., K.M. and M. Balog. A.F., J.B., K.M., J.v.d.W., N.F., B.R.-P. and P.K. advised the project. F.J.R.R. and T.L. contributed equally.

Competing interests

The authors declare no competing interests.

Additional information

Extended data is available for this paper at <https://doi.org/10.1038/s42256-025-01001-1>.

Supplementary information The online version contains supplementary material available at <https://doi.org/10.1038/s42256-025-01001-1>.

Correspondence and requests for materials should be addressed to Francisco J. R. Ruiz.






Peer review information *Nature Machine Intelligence* thanks Matías Bilkis, Florian Marquardt, Kohei Nakaji and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.

Reprints and permissions information is available at www.nature.com/reprints.

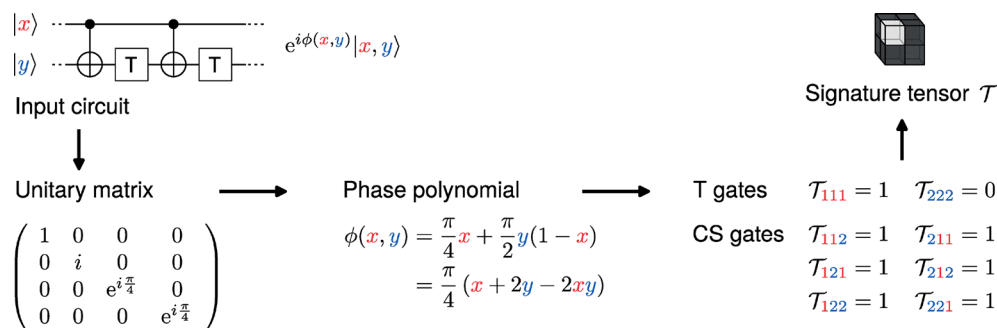
Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

© The Author(s) 2025

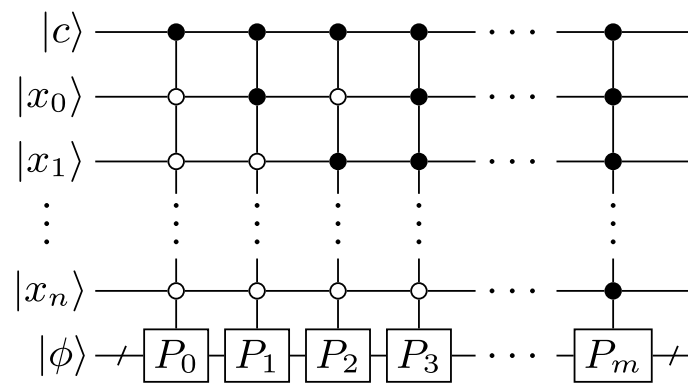
Quantum operator	#Qubits	Gate symbol	Unitary matrix
Controlled NOT (CNOT)	2		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
Controlled S (CS)	2		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{pmatrix}$
Controlled Z (CZ)	2		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$
Controlled controlled Z (CCZ)	3		$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$
Toffoli (Controlled CNOT)	3		$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$

Extended Data Fig. 1 | Common controlled gates used in quantum computing. We display the symbol of common controlled quantum gates, along with the input/output number of qubits and their corresponding unitary matrices.

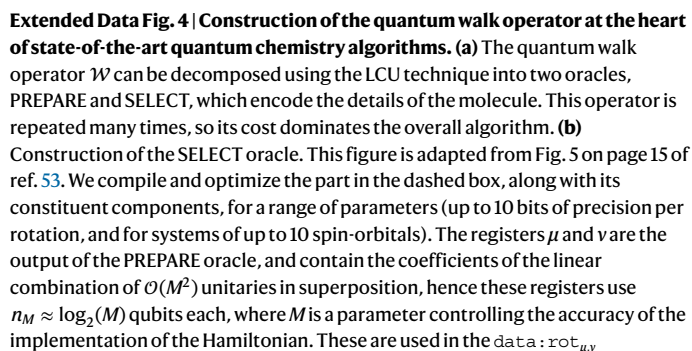


Extended Data Fig. 2 | Example of mapping a quantum circuit to a signature tensor. (Left) A circuit formed by CNOT and T gates, along with the diagonal unitary matrix that it implements. The output of the circuit is the quantum state $e^{i\phi(x,y)}|x,y\rangle$, where $x,y \in \{0,1\}$ and the phase rotation is captured by the so-called *phase polynomial* $\phi(x,y)$. **(Middle)** The phase polynomial $\phi(x,y)$ can be directly mapped to an alternative implementation of the circuit. Here, implementing the phase rotation $\phi(x,y)$ requires one T gate on the first qubit (term $\frac{\pi}{4}x$), two T gates on the second qubit (term $\frac{\pi}{2}2y$), and three CS gates (term $-\frac{2\pi}{4}xy$). Since two consecutive T gates form a (Clifford) S gate, and two consecutive CS gates form a

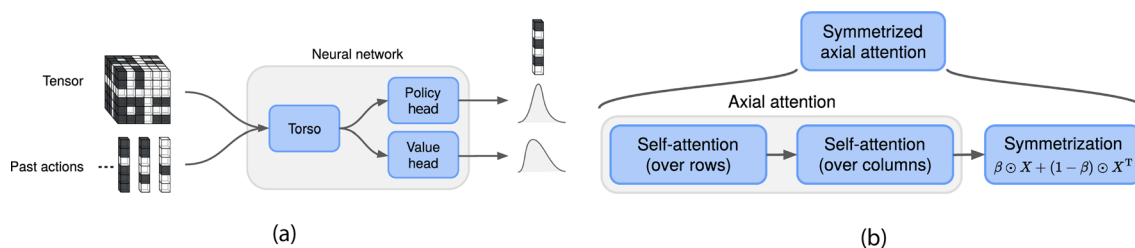
(Clifford) CZ gate, the only non-Clifford components that are needed are a T gate acting on the first qubit and a CS gate. **(Right)** The $2 \times 2 \times 2$ *signature tensor* \mathcal{T} describing the non-Clifford components. The diagonal entries of the tensor describe single-qubit interactions and correspond to the T gates. The off-diagonal entries describe two-qubit interactions and correspond to the CS gate. The signature tensor indicates a T gate acting on the first qubit and a CS gate acting on both qubits. If the circuit had more than two qubits, then entries of the tensor \mathcal{T}_{ijk} with $i \neq j, i \neq k, j \neq k$ would describe three-qubit interactions and correspond to CCZ gates.



Extended Data Fig. 3 | The action implemented by the unary iteration family of circuits. If the control qubit c is in the state $|1\rangle$, the circuit applies one of the Pauli operators from the set $\{P_1, \dots, P_m\}$ to the multiqubit state $|\phi\rangle$, depending on the value of the qubits x_1 to x_n . If the control qubit c is in the state $|0\rangle$, no operation is applied to $|\phi\rangle$.



Nature Machine Intelligence



Extended Data Fig. 5 | The neural network, featuring symmetrized axial attention layers. (a) An overview of the neural network, where the input is the current state (tensor and past actions) and the outputs are distributions over the next action to play (given by the policy head) and over the estimated return

(given by the value head). **(b)** A symmetrized axial attention layer, the building block of the neural network torso. Symmetrized axial attention incorporates a symmetrization operation after each axial attention layer to favour exchange of information between rows and columns.