scientific reports

Check for updates

OPEN

Practical overview of image classification with tensor-network quantum circuits

Diego Guala[1]✉, Shaoming Zhang[2,3], Esther Cruz[4], Carlos A. Riofrío[2], Johannes Klepsch[2] & Juan Miguel Arrazola[1]

Circuit design for quantum machine learning remains a formidable challenge. Inspired by the applications of tensor networks across different fields and their novel presence in the classical machine learning context, one proposed method to design variational circuits is to base the circuit architecture on tensor networks. Here, we comprehensively describe tensor-network quantum circuits and how to implement them in simulations. This includes leveraging circuit cutting, a technique used to evaluate circuits with more qubits than those available on current quantum devices. We then illustrate the computational requirements and possible applications by simulating various tensor-network quantum circuits with PennyLane, an open-source python library for differential programming of quantum computers. Finally, we demonstrate how to apply these circuits to increasingly complex image processing tasks, completing this overview of a flexible method to design circuits that can be applied to industrially-relevant machine learning tasks.

Tensor networks have been studied for decades across several disciplines, most notably in the context of many-body quantum systems[1,2]. In quantum computing, tensor networks have been used for the classical simulation of quantum computers[3–6] and as a framework to build new machine learning models[7–11]. Such studies have sparked interest in understanding whether tensor networks can be applied to inspire circuit design in the field of variational quantum algorithms[12–14].

Pioneering work combining tensor-network architectures and variational quantum algorithms was reported in Refs.[15–18]. The main idea is to design quantum circuits replicating tensor network architectures such as tree tensor networks and matrix product states[2,10,19]. We refer to the resulting circuits as *tensor-network quantum circuits*. One advantage of using tensor networks to design quantum circuits is that this technique provides a gradual transition from classically solvable tensor networks to ones that require a quantum computer[15]. Tensor network quantum circuits also have qubit-efficient implementations that are highly compatible with circuit cutting techniques, allowing for implementation on few-qubit machines[15,20]. Finally, any challenges encountered with these circuits benefit from the extensive literature available for classical tensor network techniques[15].

As quantum computing technologies mature, they must become accessible to a broader community of scientists, engineers, and practitioners. This is fundamental for the development of the field, as accessibility leads to more ideas that can be tested and potentially commercialized[21]. To lower the barrier of entry for practitioners interested in studying quantum tensor-network methods, this work comprehensively describes tensor-network quantum circuits and how to implement them in practice. In the Variational Tensor-Network Quantum Circuits section, we guide the reader through the process of generating quantum circuits from tensor network architectures. This includes an explanation of how to combine quantum circuit cutting techniques[20,22–27] with tensor-network quantum circuits, permitting more efficient classical simulation and providing a path for executing large circuits on devices with fewer qubits. We provide explicit formulas for the resource requirements to perform these cuts and discuss how the resources depend on the features of the original tensor network. In the Numerical Demonstrations section, we apply the above framework to perform a variety of numerical experiments. This includes benchmarks for a suite of matrix product state (MPS) quantum circuits at various bond dimensions and qubit numbers, evaluated using circuit-cutting techniques. Finally, we demonstrate applications of tensor network quantum circuits by applying them to two tasks: binary classification of simple synthetic data and defect detection in welded-metal images.

[1]Xanadu, Toronto, ON M5G 2C8, Canada. [2]BMW AG, Munich, Germany. [3]Department of Informatics, Technical University of Munich, Munich, Germany. [4]Max-Planck-Institute of Quantum Optics, Hans-Kopfermann-Straße 1, 85748 Garching, Germany. ✉email: diego@xanadu.ai

---

Variational tensor-network quantum circuits

Tensor networks. The basic building blocks of tensor networks are tensors: multi-dimensional arrays of numbers[28]. Intuitively, tensors can be interpreted as a generalization of scalars, vectors, and matrices. Consider a two-dimensional array or matrix, T. The elements of this array can be indicated by T_{ij}, where the index i indicates the rows and the index j indicates the columns of the matrix. Using the tensor network nomenclature, T is a rank two tensor. A tensor's rank is the number of indices in the tensor—a scalar has rank zero, a vector has rank one, and a matrix has rank two. While the number of dimensions of an array is equivalent to the rank of the tensor, the length in each dimension is captured by the number of possible values an index can take. This is the index dimension.

A key operation in tensor networks is *contraction*. Two tensors are contracted when they are combined into a single tensor by summing the product of their respective entries over a repeated index. For example, the standard matrix multiplication formula can be expressed as a tensor contraction

$$C_{ij} = \sum_k A_{ik} B_{kj},$$

(1)

where C_{ij} denotes the entry for the i-th row and j-th column of the product $C = AB$. Graphically, this operation can be represented as depicted in Fig. 1a. For technical and historical reasons, tensors can also be defined using indices as superscripts, e.g., the notation A_i^j can also denote a rank-2 tensor.

From tensors, we can create tensor networks. A tensor network is a collection of tensors where a subset of all indices is contracted. It is helpful to discuss tensor networks using diagrams similar to Fig. 1. In this language, tensors are represented by shapes such as circles or squares, and edges symbolizing the indices. The rule for contraction can be displayed in a tensor network diagram by connecting tensors with edges, where two connected tensors are contracted[28]. Tensor networks can represent complicated operations involving several tensors, many indices, and sophisticated contraction patterns. When multiple contractions happen in a tensor network, the corresponding summations can be performed in different orders. The sequence in which the contraction is carried out is known as the *contraction path*. This is an important concept, as a suitable contraction path will decrease the computational complexity of tensor network contraction.

Tensors of high rank can be difficult to work with since the number of array elements grows exponentially with the number of indices in the tensor. A common strategy is to express high-rank tensors as a tensor network over tensors of smaller rank. For example, consider a tensor A^{i_1,i_2,i_3,i_4} of rank four. It can be approximated by a tensor network of the form

$$A^{i_1,i_2,i_3,i_4} \approx \sum_{j_1,j_2,j_3} A_{j_1}^{i_1} A_{j_1 j_2}^{i_2} A_{j_2 j_3}^{i_3} A_{j_3}^{i_4}.$$

(2)

This tensor network is known as a matrix product state (MPS). It can be interpreted as a factorization of the tensor into a network consisting of tensors of smaller rank[2]. An MPS factorization can be used to represent tensors exactly as long as the dimension of the internal j indices, known as the *bond dimension D*, is sufficiently large. Another option is to approximate tensors with matrix product states by selecting a smaller bond dimension, which can lead to simpler computations in exchange for lower accuracy. A graphical representation of an MPS is shown in Fig. 2. Please see Refs.[2,28] for more detailed introductions to tensor networks.

Tensor-network quantum circuits. The connection between quantum computing and tensor networks can be seen by observing that quantum circuits can be expressed as tensor networks[3,5,22,30,31]. Formally, we con-
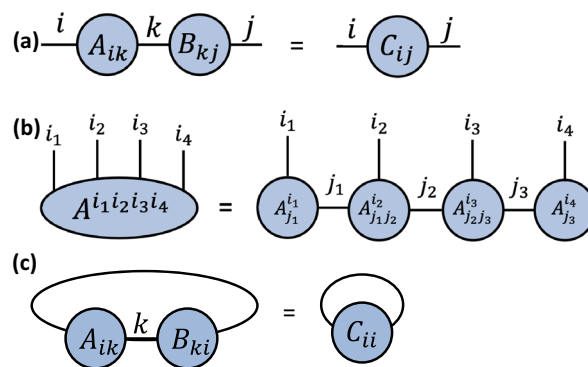

Figure 1. (**a**) A diagrammatic example illustrating the contraction of two rank-2 tensors (matrices) A and B, connected by a repeated index k. Tensors can be contracted by summing over repeated indices. In this case, the contraction corresponds to the summation over k, as given in Eq. 1; (**b**) An example of a tensor A^{i_1,i_2,i_3,i_4} and its factorization into matrix product state form $A_{j_1}^{i_1} A_{j_1 j_2}^{i_2} A_{j_2 j_3}^{i_3} A_{j_3}^{i_4}$, as given in Eq. 2. This factorization is done by singular value decomposition[29]; (**c**) An example of the trace of tensors. The trace is equal to the connection of two different legs of the tensor and summing over the corresponding index.
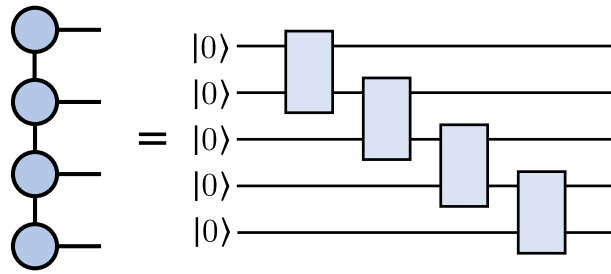
Figure 2. An example of a matrix product state (MPS) (*left*) and the corresponding quantum circuit architecture (*right*). Each tensor in the network is obtained by contracting the initial two-qubit state and a two-qubit gate.

sider a quantum algorithm consisting of an initial quantum state, represented by a density matrix ρ, a quantum circuit implementing a unitary transformation U, and a measurement performed on the output state $U\rho U^\dagger$ of the circuit, which is used to compute the expectation value $\mathrm{Tr}(U\rho U^\dagger O)$ of an observable O. This expectation value can then be expressed by the tensor network

$$A(U)^{i_1 i_2 \ldots i_n}_{j_1 j_2 \ldots j_n} A(\rho)^{j_1 j_2 \ldots j_n}_{k_1 k_2 \ldots k_n} \; A(U^\dagger)^{k_1 k_2 \ldots k_n}_{l_1 l_2 \ldots l_n} \; A(O)^{l_1 l_2 \ldots l_n}_{i_1 i_2 \ldots i_n} = \mathrm{Tr}(OU\rho U^\dagger), \qquad (3)$$

where sums are carried out implicitly over repeated indices, and $A(\rho)$, $A(U)$, $A(U^\dagger)$ and $A(O)$ are tensors representing ρ, U, $U^\dagger$ and O, respectively, as illustrated in Fig. 3.

Conversely, we can express certain tensor networks as quantum circuits, generating tensor-network quantum circuits. A tensor-network quantum circuit instructs a quantum computer to apply a transformation that is *related* to its parent tensor network, inheriting the connectivity between the tensors in the network. Since quantum circuits must apply unitary operations, we only consider mapping the tensor network elements *exactly* when the individual tensors are unitary operations. Otherwise, we allow the tensors to become general, undefined unitary operations. We refer to these resulting unitary operations as *tensor blocks*. As for the parent tensor network's bond dimension D, this is captured by the number of bond qubits, n_V, shared by each tensor block, namely as $D = 2^{n_V}$. As an illustration, Fig. 2 shows a quantum circuit with an MPS architecture, and Fig. 4 depicts a circuit following the structure of a tree tensor network (TTN).
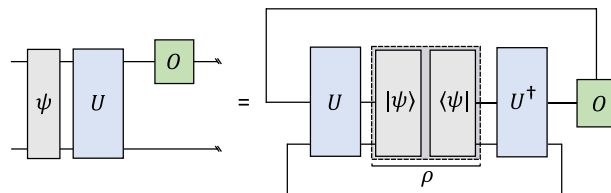

Figure 3. A quantum circuit and its corresponding tensor network. In the circuit-based picture, some initial quantum state $|\psi\rangle$ is evolved with a unitary gate U and followed by a measurement of the observable O on the top qubit (*left*). This corresponds to the contraction of a tensor network and is similar to a trace operation, as given in Eq. 3 (*right*). While other tensor network representations are possible, we choose a tensor network with bond dimension two, such that the wires in the circuit correspond to the tensor network legs.
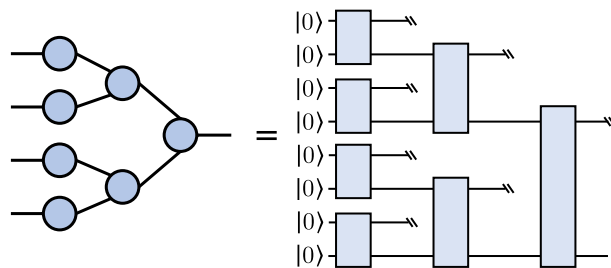

Figure 4. An example of the tree tensor network (TTN) architecture (*left*) and the corresponding quantum circuit (*right*). We use the symbol (\\) to clarify that a qubit is not affected by a particular unitary and is traced out.

A tensor-network quantum circuit is based on the shape and connectivity of its parent tensor network, for example, an MPS or TTN architecture, and not necessarily on specific tensor element values. Therefore, we can view a single tensor network architecture as a template for multiple possible circuits. We can obtain different circuits by changing the bond dimension of the parent tensor network or by varying the unitary operations corresponding to each tensor. Since each block can be an arbitrary unitary, it is crucial to define blocks that are compatible with (i) implementation on quantum hardware, (ii) fast simulation, and (iii) optimization strategies for quantum circuits. This is discussed in the following section.

Meta-ansatzes. One way to make tensor-network quantum circuits compatible with quantum hardware is to replace the unitary blocks with local circuits[18]. In this sense, just as a circuit ansatz is a strategy for arranging parametrized gates, tensor-network quantum circuits can be viewed as strategies for structuring smaller circuit ansatzes. They can therefore be interpreted as ansatzes of ansatzes, i.e., as *meta-ansatzes*. This approach allows us to employ the same techniques used to design, implement, and optimize variational quantum circuits. This is illustrated in Fig. 5, where we replace each block in a TTN circuit with a simple variational circuit. In summary, tensor networks can be used as a template to generate tensor-network meta-ansatzes. By replacing the tensor blocks in the meta-ansatz with parameterized circuits, we obtain a final variational quantum circuit. This circuit can then be simulated, implemented on hardware, and optimized as any other variational circuit.

Designing tensor-network quantum circuits. There are multiple ways to generate quantum circuits that relate to parent tensor networks[15–18,32–34]. We do not aim to rigorously reproduce the tensor network contraction with a quantum circuit, and instead endeavor to preserve certain features of the original tensor network: the number of tensors or operations, the connectivity between the operations, and the bond dimension of the connections. We now describe a procedure to generate tensor-network quantum circuits that maintain these features.

Consider a tensor network $T = (G, A)$ consisting of a collection of tensors $A = \{a(v) : v \in V\}$ and an undirected graph $G(E, V)$ defined by a set of edges E and a set of vertices V, that admits open edges[22,35]. The vertices V of G represent tensors, whereas the tensor indices are depicted by the edges E of G. For a tensor network $T = (G(E, V), A)$, we can construct a quantum circuit $C = G_c(E_c, V_c)$ such that $E \subseteq E_c, V = V_c$. A valid quantum circuit graph must be acyclic and have an equal number of incoming and outgoing open edges, representing the wires in the circuit. An index with dimension d_e in the tensor network is replaced with $\lceil \log_2 d_e \rceil$ qubit wires in the corresponding circuit. In other words, we can create a directed acyclic graph, as required for quantum circuits, while preserving the connections and number of vertices in the original tensor network. Below we outline the full procedure in detail.

1. Choose the direction of the open edges in the tensor network. As in Ref.[15], this is informed by the desired application of the quantum circuit. For example, a generative machine learning task may use the open edges as outputs while a discriminative task may use them as inputs[15].
2. Label the vertices with integers, v, in increasing order such that no two vertices have the same label.
3. Set every edge (v_i, v_j) to point from the lower integer to the higher integer vertex such that $v_i < v_j$. This ensures the graph is acyclic.
4. Add new open edges to the circuit graph G_c: for each vertex where the number of incoming edges, n_{in}, is different from the number of outgoing edges, n_{out}, we add new directed edges such that $n_{in} = n_{out}$. This addresses the requirement that circuit operations must have equal numbers of incoming and outgoing wires.
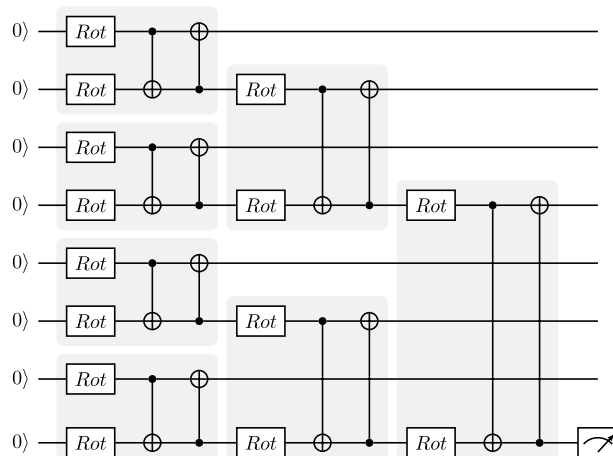

Figure 5. A tensor-network circuit as a meta-ansatz. We employ the same architecture as in Fig. 4 and replace the tensor blocks with variational circuits consisting of single-qubit rotations and CNOT gates.

The resulting graph represents a quantum circuit where the vertices are unitary operations and the edges are qubits.

As an example, Fig. 6 shows how we can obtain a quantum circuit graph from a projected entangled pair states (PEPS) tensor network graph. In Fig. 6, the tensor network starts with nine tensors and nine open edges. This evolves into a quantum circuit with ten qubits and nine unitary operations. It is important to note that the number of qubits connecting two gates corresponds to the dimension of the index between the related tensors. While the example assumes a bond dimension of two for the parent tensor network, we can account for an increase in bond dimension by duplicating wires in the quantum circuit, such that $D = 2^{n_V}$.

Circuit cutting.   A quantum circuit can be executed on hardware or simulated classically, but hybrid methods also exist that trade off classical and quantum resources[20,22,24]. One of these techniques, circuit cutting[20,22], enables the execution of many-qubit circuits with few-qubit quantum devices, albeit at the expense of additional classical computation. The primary strategy in this technique is to divide large circuits into smaller fragments which are then evaluated on fewer-qubit devices. By evaluating these fragments over a large number of different configurations, we can obtain enough results to classically reconstruct the output of the original circuit.

More generally, recall that a quantum circuit can be described by a directed acyclic graph $G_c(V_c, E_c)$, where the nodes represent gates in the circuit and the edges represent wires. As we now explain, cutting a circuit is linked to partitioning this circuit graph. A partitioning Π of a graph is a collection of subsets of vertices $V_1, V_2, \ldots, V_k \subset V_c$ such that every vertex in the graph is contained in exactly one subset. We refer to each subset V_i as a graph fragment. The edges connecting different graph fragments correspond to wires that can be cut in the procedure, producing circuit fragments that can be executed separately. This is summarized in Fig. 7.

The graph-based framework described above can be used to analyze the resource requirements of circuit cutting. Ref.[22] shows that the number of circuit executions needed to compute the expectation value of a tensor product of local observables of the form $O = \bigotimes_{i=1}^{n} O_i$ with precision ε scales asymptotically as

$$O\left(8^{3d_{\max}} d_{\max}(k^3 \log k)/\epsilon^2\right), \tag{4}$$

where k is the number of fragments and $d_{\max}$ is the maximum number of edges between fragments. For the case of MPS circuits, this cost may be quadratically reduced following the techniques of Ref.[23].

Looking into Eq. (4), we find that tensor-network quantum circuits are naturally suited for circuit cutting techniques: these circuits can be cut such that each tensor block results in a fragment and the exponent d_{max} is kept fixed, allowing the circuit to be executed on few qubits while the number of circuits to evaluate scales polynomially with respect to the number of tensor blocks. For example, for architectures like MPS and TTN,
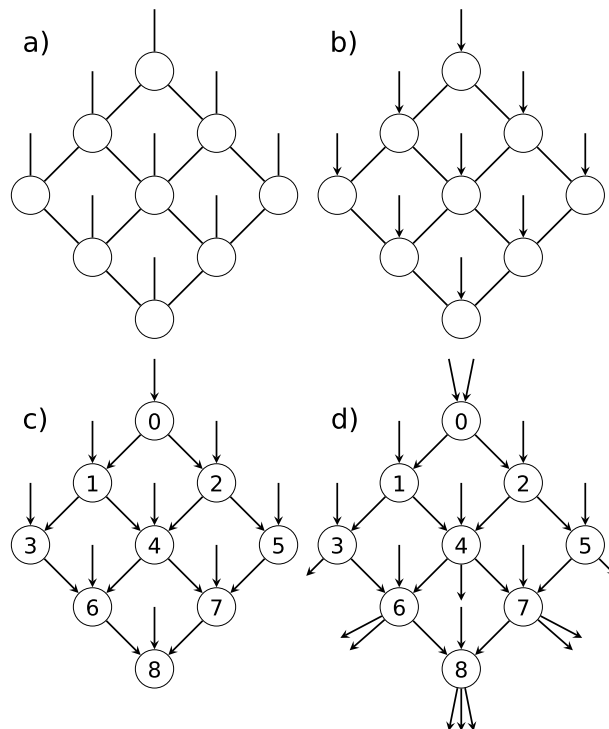

Figure 6. (**a**) An example PEPS tensor network with bond dimension two, (**b**) tensor-network quantum circuit design step 1: adding direction to the open edges in the tensor network diagram, (**c**) steps 2 and 3: labeling the vertices and adding acyclic direction to the internal edges, (**d**) step 4: adding edges to balance the in- and out-degree of each vertex.
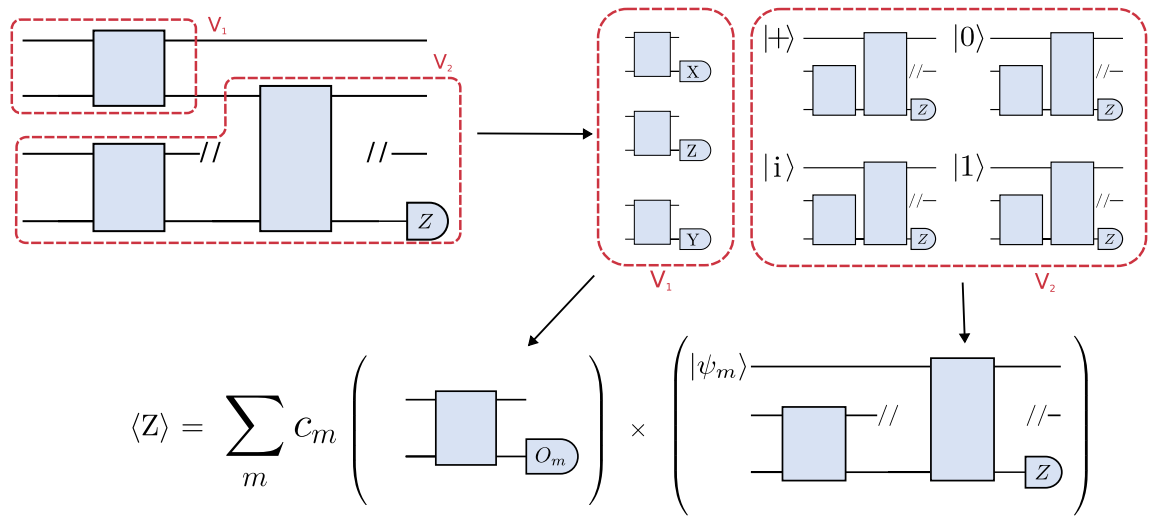
Figure 7. Circuit cutting and reconstruction procedure for a small TTN-shaped quantum circuit. *Top*: the original circuit is partitioned into two fragments, V_1 and V_2. One fragment is executed with multiple different measurements, $\langle O_m \rangle$, while the other fragment is executed with multiple different initial states, $|\psi_m\rangle$. *Bottom*: the results of the fragment executions are combined as dictated by Refs.[20,22]. This summation is performed with a classical computer and returns the expectation value of a measurement on the original circuit.

the maximum number of edges between fragments $d_{\max}$ is equal to the number of bond qubits, n_V, that connect two adjacent blocks. Since n_V is chosen in the design, it is possible to increase the number of qubits in the circuit while keeping n_V constant. With a constant n_V, k only increases linearly with respect to the total number of qubits. This means we can extend tensor-network quantum circuits as in Fig. 7 to more qubits and deeper circuits as in Fig. 5, while the number of quantum circuit fragments we have to evaluate only increases polynomially with the number of qubits.

More precisely, consider an MPS circuit with bond dimension 2^{n_V}, with blocks of $2n_V$ qubits, defined on n total qubits and with a single Pauli Z measurement on the bottom qubit. We can cut the circuit into its constituent blocks, meaning that we can evaluate the full circuit on a device with only $2n_V$ qubits. In this case, the number of different circuits that must be evaluated to reconstruct the measurement on the original circuit is given by:

$$c_{\text{MPS}} = 3^{n_V} + \left(\frac{n}{n_V} - 3 \right) 4^{n_V} 3^{n_V} + 4^{n_V}. \tag{5}$$

For a TTN circuit with bond dimension 2^{n_V}, with blocks of $2n_V$ qubits, defined on n total qubits and with a single Pauli Z measurement on the bottom qubit, this becomes:

$$c_{\text{TTN}} = \frac{3^{n_V} n}{2n_V} + 3^{n_V} (4^{2n_V}) \left(\frac{n}{2n_V} - 2 \right) + 4^{2n_V}. \tag{6}$$

Numerical demonstrations

To illustrate possible applications of tensor-network quantum circuits, we perform a series of numerical simulations. We combine circuit cutting techniques with tensor network circuits to classify synthetic data and then extend the model to image classification and object detection on industrial data. To carry out these experiments, we build on the open-source PennyLane library for quantum differentiable programming[36].

Circuit cutting simulation times.   In this section, we start by benchmarking the runtime performance for the combination of circuit cutting techniques with tensor-network quantum circuits. We show how simulation time increases as we scale various tensor-network parameters.

More specifically, we design an MPS quantum circuit as in Fig. 2, where the unitary blocks are replaced with two strongly entangling layers[37] like the ones in Fig. 8. An example of a resulting MPS with the unitary blocks specified as strongly entangling layers is given in Fig. 9 for four qubits. We then add a Pauli Z measurement on the bottom qubit and simulate the resulting circuit. PennyLane's SPSVERBc1 template can be used to produce an MPS circuit with user-defined circuit blocks, number of bond qubits, and total number of qubits. By defining a block that includes the strongly entangling layers template, we can define a circuit like in Fig. 9. We then use PennyLane's circuit cutting functionality to separate the circuit into its individual tensor blocks, add the required state preparations and observables, evaluate them, and reconstruct the original circuit result. This is done automatically when the SPSVERBc2 decorator is applied to a PennyLane circuit. The simulations are performed for various configurations of the bond dimension, block size, and the total number of qubits. For examples of how to use PennyLane to simulate circuit cutting, please check the PennyLane documentation.
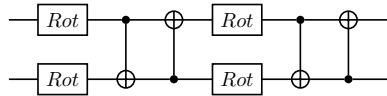
Figure 8. Two strongly entangling layers for two qubits. Each set of two rotation gates and two CNOT gates constitutes one strongly entangling layer. These layers can be repeated any number of times and extended to any number of qubits[37].
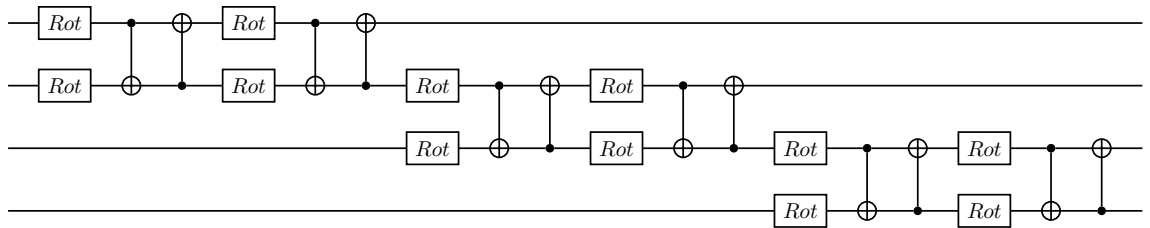

Figure 9. MPS meta-ansatz with two strongly entangling layers replacing each unitary tensor block.

According to Eqs. 4–6, we find that the simulation time increases polynomially with the total number of qubits and exponentially with the number of bond qubits. This is shown in Fig. 10. Overall, applying circuit cutting to an MPS quantum circuit enables the simulation of a large number of qubits as long as the number of bond qubits is kept low. For the simple structure of these circuits, this performance could also be achieved with simulations based on classical tensor network techniques[3], but circuit cutting provides a path toward executing large circuits using small quantum computers.

Bars and stripes. Here we demonstrate how to use a tensor-network quantum circuit to perform image classification tasks. The problems we study are well-known and can be routinely solved with classical methods. Our purpose is not to compete with such techniques, but rather to guide readers on example applications of tensor-network quantum circuits.

The bars and stripes data set is an example of synthetic data often used to develop proof-of-principle machine learning algorithms. As shown in Fig. 11, a bars and stripes instance is composed of binary black and white images of size $n \times n$ pixels, where either all pixels in a column have the same color (bars) or all pixels in a row have the same color (stripes)[38]. The classification task is to output the correct label, bars or stripes, for any input image from the data set. To perform this task, we implement a quantum circuit consisting of an encoding
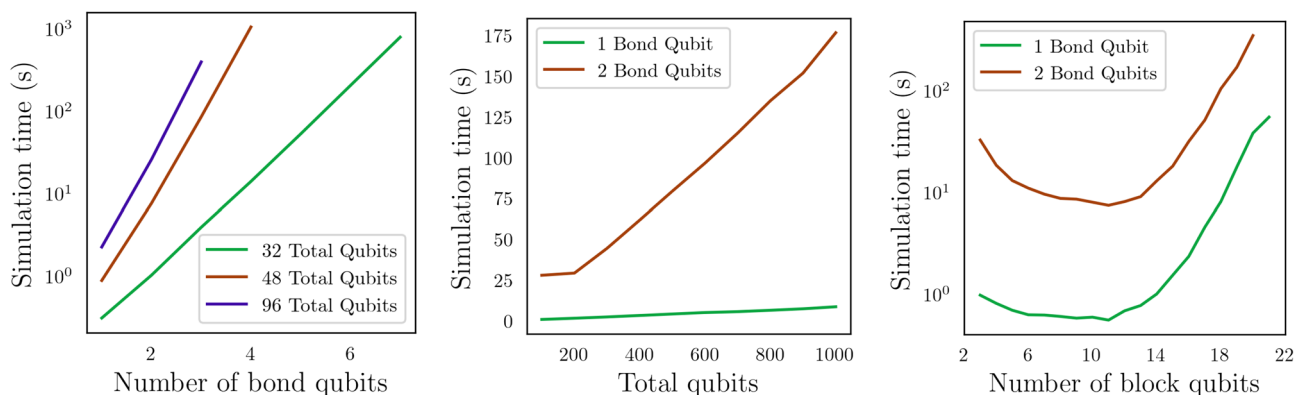

Figure 10. *Left*: The simulation time of an MPS circuit increases exponentially with the number of bond qubits, regardless of the total number of qubits. For this data, we used 16-qubit blocks. Note that the total number of qubits can vary slightly as the number of bond qubits changes. The MPS shape dictates the variation in qubit numbers, e.g., an MPS circuit with 16-qubit blocks and two bond qubits per block can only result in circuits with $16 + 14n$ qubits, where n is a positive integer. *Middle*: At a constant number of bond qubits and five block qubits, the simulation time increases linearly with the total number of qubits. *Right*: For a constant total circuit size of 100 qubits, increasing the size of the tensor blocks initially reduces the simulation time and then increases it. This is an artifact of how the tensor blocks are defined. Initially, increasing the number of block qubits reduces the total number of circuits to simulate during circuit cutting. However, as the size of the blocks increases, the time gained by having larger circuits surpasses the time saved by having fewer circuits. All simulations are performed on a personal laptop computer with 16 GB of RAM and a four-core i7-1185G7 processor operating at 3.00 GHz.
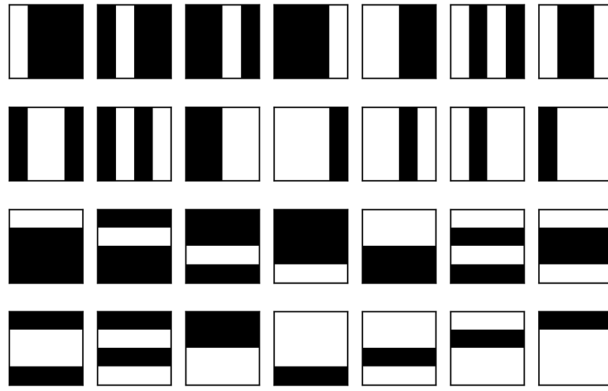
Figure 11. Bars and stripes data set for 4×4 pixel images.

operation to input the image, a parameterized tensor-network quantum circuit to process it, and a measurement to obtain the label. Since many design choices are required to implement this framework, we summarize these in the following list:

1. We choose the amplitude encoding[39] to encode the normalized datapoint x of pixel information into the amplitude of an n-qubit quantum state $|\psi_x\rangle = \sum_{i=1}^{2^n} x_i |i\rangle$, with x_i the i-th element of x and $|i\rangle$ the i-th computational basis state.
2. We choose a tree tensor network architecture because its hierarchical structure is suited to perform image processing tasks like convolution and pooling[15,40,41].
3. We use two strongly entangling layers to replace the unitary blocks in the circuit because they are expressive[37] and experimentation showed that two layers can reach 100% classification accuracy for this application.
4. We limit the individual blocks to two qubits, to reduce computation time while still reaching 100% training accuracy.
5. We make a Pauli Z measurement on the bottom qubit to obtain the labels. When an input image results in a Pauli Z measurement of positive one, we label that image "bars" and when the Pauli Z measurement is negative one, we label the image "stripes". For multiple measurements, we use the expectation value, such that when $\langle \sigma^Z \rangle > 0$, we label the image "bars" and when $\langle \sigma^Z \rangle < 0$, we label the image "stripes". In other words, we choose the most-frequently-sampled label.
6. We choose 14 training images and 14 test images from the bars and stripes data set.
7. We use the loss function

$$loss = \sum_i \left(1 + 10 e^{7p_i}\right)^{-1},$$
(7)

where the index i iterates over the images in the data set, and p_i is the probability of obtaining the correct label when sampling the circuit with image i as input. This loss function favors a good probability of sampling correct labels over many images rather than a very high probability over a few images. The parameter p_i can be calculated from the Pauli Z expectation value as:

$$p_i = 1 - \left| \frac{1 - \langle \phi_i | \sigma_n^z | \phi_i \rangle}{2} - \ell \right|,$$
(8)

Where σ_n^z is the Pauli Z operator applied to the n-th qubit, $|\phi_i\rangle$ is the final state of the qubits after running the circuit for image i, and $\ell = 0, 1$ is the correct image label, taking a value of zero for bars and one for stripes.
8. We use the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm to optimize the circuit, with hyperparameters $\alpha = 0.602$ and $\gamma = 0.101$[42].
9. We use PennyLane[36] templates to design the tensor-network circuits and Jet[43] to simulate the circuits using optimized task-based tensor contraction.

An example circuit on four qubits following these design choices is shown in Fig. 12.

Next, we train the tree tensor network circuit on images of size 4×4 pixels, then extend to 16×16 pixels, and finally reach 256×256 pixels. Under these design conditions, we can train the circuit to reach 100% classification accuracy for both the training and test sets. This is most likely due to the simplicity of the task, as we will see in the next section. These results are summarized in Fig. 13.

Welding defects. In this section, we extend the previous image classifier circuit to perform object detection on weld images toward implementing quality control systems[44]. Welding is a standard method to fuse two portions of metallic material. It consists of partially melting the metal to attach the materials and allowing it to solidify. During this process, defects can weaken the connection between the materials. The welding defects data set contains cross-sectional X-ray images of the fused portion in different welded structures. The flaws in
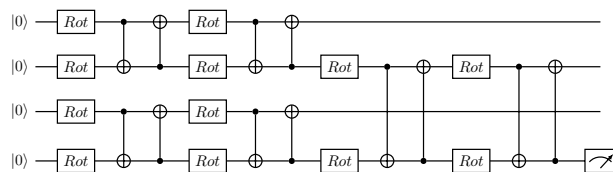
Figure 12. A tree tensor quantum circuit with $n_V = 1$ and two entangling layers applied to four qubits. Single-qubit gates apply arbitrary Bloch rotations of user-defined value ω in the Z axis, θ in the Y axis, and ϕ in the X axis. These rotations can be optimized such that the circuit classifies 16×16 pixel images. The circuit can also be extended to more qubits, enabling the classification of larger images.
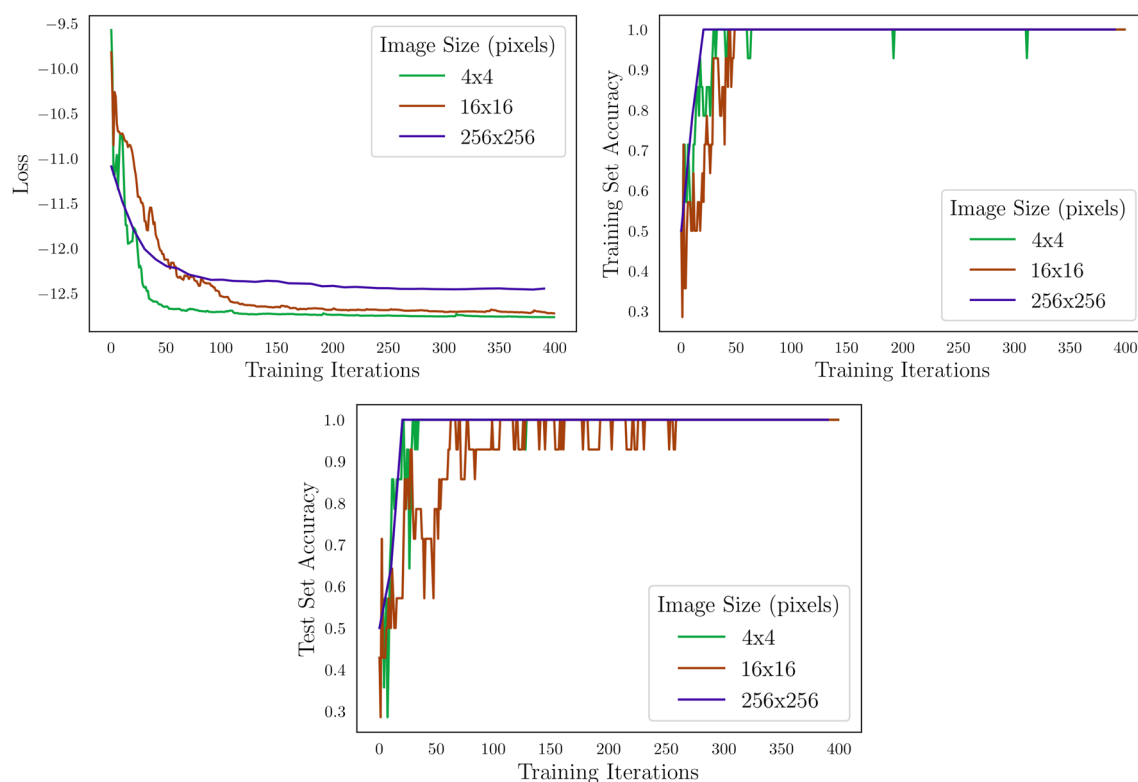

Figure 13. Evolution of loss, training accuracy, and test accuracy while training a tree tensor quantum circuit on various image sizes. The quantum circuit reaches 100% accuracy for all sizes within 400 training iterations.

the images appear as very dark or black cracks and bubbles, as seen in Fig. 14. The goal here is to determine the severity and extent of the defects by quantifying the area they occupy using a tree tensor network circuit that identifies the size and location of the defect.

This object detection is performed in three steps. We first classify an image as containing defects or not. We then use the sliding windows method[45] to segment the image into many smaller pictures and classify each segment with 16×16 pixel segments to propose defect areas. In the final step, we re-segment the proposed areas into 4×4 pixel images to identify the individual defects in the proposed areas. This strategy requires training three image classifiers, one for each step. The overall defects-versus-no-defects classifier is a 16-qubit tree tensor network circuit. To train this classifier, we manually select 14 defect and 14 non-defect images and divide them into a training set and test set. We crop and resize the images to 256×256 pixels before inputting them to the circuit. This is done both to fit the 16-qubit size requirement and because the defects are typically in the center of the images.

For the second classifier, we use an eight-qubit circuit that can process the 16×16 pixel segments. To train this circuit, we segment an image with defects into 16×16 pixel images and manually select 14 segments with defects and 14 without defects. Finally, we repeat the previous procedure with a four-qubit circuit and 4×4 pixel images. Due to the smaller size, the four-qubit and eight-qubit circuits are significantly faster to train and simulate than the 16-qubit circuit.

In summary, the complete defect detection strategy involves first running a 16-qubit tree tensor circuit to classify whether the center 256×256 pixel portion of the weld image has a flaw, running the sliding window algorithm to classify 16×16 pixel segments of the image to propose sections with flaws, and finally running the sliding window algorithm to classify 4×4 pixel segments of the proposed area. Once the final 4×4 pixel
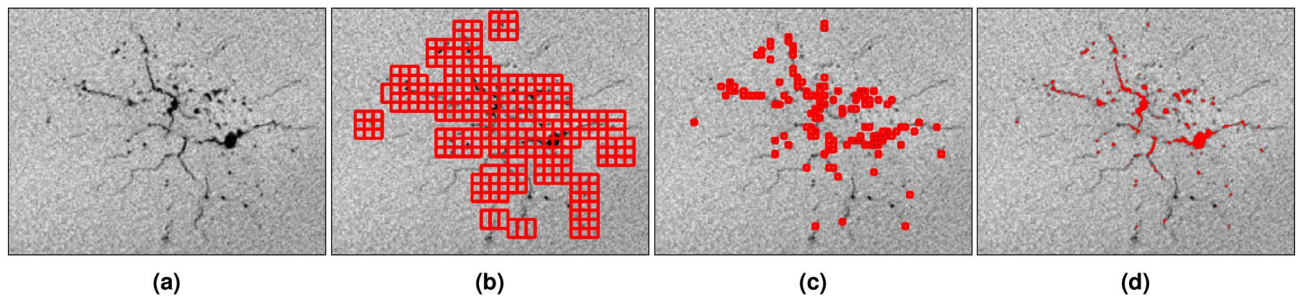
(a)   **(b)**   **(c)**   **(d)**

Figure 14. (**a**) Welding defects example image, followed by (**b**) coarse-grained bounding boxes, (**c**) fine-grained bounding boxes, and (**d**) highlighted defects. This figure illustrates the full defect detection procedure and the results at each step. Image (**a**) is identified as having defects by a 256×256 pixel classifier. This classifier is trained on manually chosen images, obtaining 79% accuracy on a training set and 71% accuracy on a test set. After segmenting the original image (**a**) into 16×16 pixel images, classifying each segment, and highlighting the defective segments, we obtain image (**b**). This classifier is trained on manually chosen images, obtaining 100% accuracy on the training set and 79% accuracy on the test set. Further segmenting these highlighted sections into 4×4 pixel images and classifying yields image (**c**). This classifier reaches 100% accuracy on a similarly defined training set and 71% accuracy on a test set. To visualize the percentage of the defect that is captured, we classically post-process the highlighted sections, converting black pixels in these areas to red and showing the result in image (**d**). While the algorithm occasionally misses small defects, the more severe portions are identified.

segments are classified, we can classically select the black pixels in those segments and convert them to red to highlight the detected defect area. The results of running the entire algorithm on an example image are shown in Fig. 14. This figure demonstrates that the procedure indeed detects the defect areas.

Conclusion

In this work, we have provided an overview of how to apply tensor-network architectures to the design of variational quantum circuits. We implement these variational circuits to address illustrative industry-relevant problems. The results serve as examples of potential proof-of-principle use cases for existing quantum hardware and simulators. Additionally, the results show how combining circuit cutting with tensor-network quantum circuits can improve the scale of quantum systems that can be simulated in classical computers. Additionally, the results can be leveraged to execute large tensor-network quantum circuits on small quantum devices. Moreover, we find that simple image-classification tasks can be performed on quantum computers via this method.

Additional work must be done comparing the performance of tensor network quantum circuits to classical alternatives. While we do not anticipate that tensor-network quantum circuits will outperform classical algorithms for the investigated image-processing applications, the tensor-network quantum circuit framework may help study the relationships between data structure and the design of quantum algorithms.

Data availability

The datasets generated during and/or analysed during the current study are not publicly available due to corporate privacy policy of BMW Group but are available from the corresponding author on reasonable request. A demonstration is provided to explain how to use PennyLane templates to design and implement tensor-network quantum circuits discussed in this paper[46].

References
1. Cirac, J. I., Pérez-García, D., Schuch, N. & Verstraete, F. Matrix product states and projected entangled pair states: Concepts, symmetries, theorems. *Rev. Mod. Phys.* **93**, 045003. https://doi.org/10.1103/RevModPhys.93.045003 (2021).
2. Orús, R. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Ann. Phys.* **349**, 117–158. https://doi.org/10.1016/j.aop.2014.06.013 (2014).
3. Markov, I. L. & Shi, Y. Simulating quantum computation by contracting tensor networks. *SIAM J. Comput.* **38**, 963–981 (2008).
4. Zhou, Y., Stoudenmire, E. M. & Waintal, X. What limits the simulation of quantum computers?. *Phys. Rev. X* **10**, 041038 (2020).
5. Huang, C. et al. Classical simulation of quantum supremacy circuits. arXiv:2005.06787 (2020).
6. Pan, F., Chen, K. & Zhang, P. Solving the sampling problem of the sycamore quantum supremacy circuits. arXiv:2111.03011 (2021).
7. Stoudenmire, E. M. & Schwab, D. J. Supervised learning with quantum-inspired tensor networks (2017). arXiv:1605.05775.
8. Martyn, J., Vidal, G., Roberts, C. & Leichenauer, S. Entanglement and tensor networks for supervised image classification. arXiv:2007.06082 (2020).
9. Han, Z.-Y., Wang, J., Fan, H., Wang, L. & Zhang, P. Unsupervised generative modeling using matrix product states. *Phys. Rev. X* **8**, 031012 (2018).
10. Wall, M. L. & D'Aguanno, G. Tree-tensor-network classifiers for machine learning: From quantum inspired to quantum assisted. *Phys. Rev. A* **104**, 042408 (2021).
11. Grant, E. *et al.* Hierarchical quantum classifiers. *npj Quantum Inf.* **4**, 1–8 (2018).
12. Cerezo, M. *et al.* Variational quantum algorithms. *Nat. Rev. Phys.* **3**, 625–644 (2021).

13. Schuld, M., Sinayskiy, I. & Petruccione, F. An introduction to quantum machine learning. *Contemp. Phys.* **56**, 172–185 (2015).
14. Wright, L. et al. Deterministic tensor network classifiers. arXiv preprint arXiv:2205.09768 (2022).
15. Huggins, W., Patil, P., Mitchell, B., Whaley, K. B. & Stoudenmire, E. M. Towards quantum machine learning with tensor networks. *Quantum Sci. Technol.* **4**, 024001. https://doi.org/10.1088/2058-9565/aaea94 (2019).
16. Cong, I., Choi, S. & Lukin, M. D. Quantum convolutional neural networks. *Nat. Phys.* **15**, 1273–1278 (2019).
17. Haghshenas, R. Optimization schemes for unitary tensor-network circuit. *Phys. Rev. Res.* **3**, 023148 (2021).
18. Haghshenas, R., Gray, J., Potter, A. C. & Chan, G.K.-L. Variational power of quantum circuit tensor networks. *Phys. Rev. X* **12**, 011047 (2022).
19. Tagliacozzo, L., Evenbly, G. & Vidal, G. Simulation of two-dimensional quantum systems using a tree tensor network that exploits the entropic area law. *Phys. Rev. B* **80**, 235127 (2009).
20. Tang, W., Tomesh, T., Suchara, M., Larson, J. & Martonosi, M. Cutqc: Using small quantum computers for large quantum circuit evaluations. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* 473–486 (2021).
21. Bayerstadler, A. *et al.* Industry quantum computing applications. *EPJ Quantum Technol.* **8**, 25. https://doi.org/10.1140/epjqt/s40507-021-00114-x (2021).
22. Peng, T., Harrow, A. W., Ozols, M. & Wu, X. Simulating large quantum circuits on a small quantum computer. *Phys. Rev. Lett.* **125**, 150504 (2020).
23. Lowe, A. *et al.* Fast quantum circuit cutting with randomized measurements. https://doi.org/10.48550/ARXIV.2207.14734 (2022).
24. Bravyi, S., Smith, G. & Smolin, J. A. Trading classical and quantum computational resources. *Phys. Rev. X* **6**, 021043 (2016).
25. Piveteau, C. & Sutter, D. Circuit knitting with classical communication. arXiv preprint arXiv:2205.00016 (2022).
26. Perlin, M. A., Saleem, Z. H., Suchara, M. & Osborn, J. C. Quantum circuit cutting with maximum-likelihood tomography. *npj Quantum Inf.* **7**, 1–8 (2021).
27. Dunjko, V., Ge, Y. & Cirac, J. I. Computational speedups using small quantum devices. *Phys. Rev. Lett.* **121**, 250501 (2018).
28. Biamonte, J. & Bergholm, V. Tensor networks in a nutshell (2017). arXiv:1708.00006.
29. Schollwöck, U. The density-matrix renormalization group in the age of matrix product states. *Ann. Phys.* **326**, 96–192. https://doi.org/10.1016/j.aop.2010.09.012 (2011).
30. Shi, Y.-Y., Duan, L.-M. & Vidal, G. Classical simulation of quantum many-body systems with a tree tensor network. *Phys. Rev. A* **74**, 022320 (2006).
31. Arad, I. & Landau, Z. Quantum computation and the evaluation of tensor networks. *SIAM J. Comput.* **39**, 3089–3121 (2010).
32. Ran, S.-J. Encoding of matrix product states into quantum circuits of one- and two-qubit gates. *Phys. Rev. A* https://doi.org/10.1103/physreva.101.032310 *(2020)*.
33. Rudolph, M. S., Miller, J., Chen, J., Acharya, A. & Perdomo-Ortiz, A. Synergy between quantum circuits and tensor networks: Short-cutting the race to practical quantum advantage. https://doi.org/10.48550/ARXIV.2208.13673 (2022).
34. Rudolph, M. S., Chen, J., Miller, J., Acharya, A. & Perdomo-Ortiz, A. Decomposition of matrix product states into shallow quantum circuits. https://doi.org/10.48550/ARXIV.2209.00595 (2022).
35. Robeva, E. & Seigal, A. Duality of graphical models and tensor networks. *Inf. Inference J. IMA* **8**, 273–288 (2019).
36. Bergholm, V. et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations (2020). arXiv:1811.04968.
37. Schuld, M., Bocharov, A., Svore, K. M. & Wiebe, N. Circuit-centric quantum classifiers. *Phys. Rev. A* https://doi.org/10.1103/physreva.101.032308 *(2020)*.
38. Benedetti, M. *et al.* A generative modeling approach for benchmarking and training shallow quantum circuits. *npj Quantum Inf.* **5**, 45. https://doi.org/10.1038/s41534-019-0157-8 (2019).
39. Schuld, M. & Petruccione, F. Machine Learning with Quantum Computers (Springer, 2021).
40. Liu, D. *et al.* Machine learning by unitary tensor network of hierarchical tree structure. *New J. Phys.* **21**, 073059. https://doi.org/10.1088/1367-2630/ab31ef (2019).
41. Cohen, N., Sharir, O. & Shashua, A. On the expressive power of deep learning: A tensor analysis (2016). arXiv:1509.05009.
42. Spall, J. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Trans. Aerosp. Electron. Syst.* **34**, 817–823. https://doi.org/10.1109/7.705889 (1998).
43. Vincent, T. et al. Jet: Fast quantum circuit simulations with parallel task-based tensor-network contraction (2021). arXiv:2107.09793.
44. Guijo, D. et al. Quantum artificial vision for defect detection in manufacturing. arXiv preprint arXiv:2208.04988 (2022).
45. Dalal, N. & Triggs, B. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 886–893 (IEEE, 2005).
46. Guala, D., Cruz-Rico, E., Zhang, S. & Arrazola, J. M. Pennylane tutorial: Tensor-network quantum circuits (2022).

Acknowledgements

Author contributions

All authors contributed to the study meaningfully. D.G. wrote the main manuscript text and conceptualized and conducted numerical experiments with S.Z. and E.C., while C.A.R., J.K. and J.M.A. are the main coordinators of the project and contributed to the conceptualization and funding acquisition. All authors contributed to writing-review and editing.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to D.G.

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.