Representation of binary classification trees with binary features by quantum circuits

Raoul Heese[1], Patricia Bickert[1], and Astrid Elisa Niederle[2]

[1]Fraunhofer ITWM, 67663 Kaiserslautern, Germany
[2]BASF SE, 67063 Ludwigshafen, Germany

We propose a quantum representation of binary classification trees with binary features based on a probabilistic approach. By using the quantum computer as a processor for probability distributions, a probabilistic traversal of the decision tree can be realized via measurements of a quantum circuit. We describe how tree inductions and the prediction of class labels of query data can be integrated into this framework. An on-demand sampling method enables predictions with a constant number of classical memory slots, independent of the tree depth. We experimentally study our approach using both a quantum computing simulator and actual IBM quantum hardware. To our knowledge, this is the first realization of a decision tree classifier on a quantum device.

1 Introduction

Decision trees are well-known predictive models commonly used in data mining and machine learning for a wide area of applications [1–3]. In general, a decision tree can be viewed as a flowchart-like structure that can be used to query data. Starting from the root, each internal node represents a test on the query data and each outgoing branch represents a possible outcome of this test. For a binary tree, the test result is a Boolean value and can therefore be either true or false (i. e., there are two branches from each internal node). Each leaf of the tree can be associated with a decision. Therefore, the path from the root to the leaf implies a set of decision rules for the query data in the sense of a sequential decision process. In particular, we consider *binary classification trees* where the decision of a leaf determines the membership of a data point to a predefined discrete set of classes.

The inference of a decision tree from a given data set is a supervised machine learning task also known as decision tree *induction* (or decision tree learning). However, finding a globally optimal solution is NP-hard [4, 5] and therefore heuristic recursion algorithms are favored in practice [6]. Such algorithms typically work in a greedy top-down fashion [7]: Starting from the root, the best test is estimated for each internal node by minimizing a data impurity function. The data set is splitted accordingly into two subsets along each of the two outgoing branches. This process is repeated recursively for each internal node until a stopping criterion terminates the tree traversal and results in a leaf with a classification decision based on the majority class present in the data subset within the node. The algorithm ends when all paths lead to a leaf. A heuristically created decision tree is not guaranteed to be globally optimal but might still be suitable for practical purposes.

In the context of quantum computing, decision trees can be assigned to the field of *quantum machine learning* [8]. Several previous papers consider the interplay between decision trees and quantum computing. In [9], the traversal speed of decision trees is studied and classical and quantum approaches are compared. The authors find no benefit of one over the other. An heuristic algorithm to induce quantum classification trees is presented in [10], where data points are encoded as quantum states and measurements are used to find the best splits. However, some parts of the

Raoul Heese: raoul.heese@itwm.fraunhofer.de

hybrid quantum-classical algorithm seem to be incomplete and can be considered "enigmatic" [11]. In [12], a conceptional quantum extension is presented for the classical decision tree induction algorithm *C5.0* [2] such that an almost quadratic speed-up can be gained by implementing a suitable quantum subroutine. However, there is no publication that we know of that considers both induction and evaluation of a quantum decision tree in a consistent manner. In this sense, "the potential of a quantum decision tree is still to be established" [11].

Decision trees have also been used to realize efficient data management strategies in a quantum computing context. In [13], a "bucket-brigade" quantum random access memory (qRAM) is proposed, which achieves an exponential reduction in access overhead compared to a conventional classical random access memory design. In this approach, three-level memory elements are structured in a binary tree to organize the flow of information to the memory cells in the leaves (and back). Moreover, a whole ensemble of decision trees can also be used to store structured data. In [14], binary decision trees are used as building blocks for a data structure to store a preference matrix for a quantum recommendation system. To this end, the preference matrix is decomposed into row vectors, which are then stored in a binary tree such that each leaf holds the amplitude of one vector component and each internal node stores the respective amplitude sums of its branches, which enables efficient computations.

The term "quantum decision tree" is also used in the context of query complexity analysis of quantum algorithms in analogy to the decision tree view of classical query complexity. Here, the quantum algorithm is treated as a sequence of unitary operators followed by a measurement [15–18]. However, such a configuration has a different purpose than a decision tree in the sense of a predictive model, which is why the term "quantum query algorithm" is also used instead. On the other hand, a classical decision tree query can in principle be expressed by a quantum query algorithm.

In this manuscript, we follow a different strategy and propose a quantum representation of decision trees, which we call *Q-tree*, by using the quantum computer as a processor for probability distributions. For this purpose, we first take a probabilistic perspective to describe the decisions in a (classical) decision tree as conditions of the probability distribution of the training data. This approach allows us to associate a marginalized conditional probability distribution to each leaf that describes the probability distribution to reach the leaf in a probabilistic traversal of the tree and the corresponding class label probability distribution, respectively. In this sense, we *load* a classical decision tree into a quantum circuit.

The set of conditions in the tree can also be parameterized to obtain a tuple of vectors that uniquely define the structure of the tree. Based on these parameters, an appropriate quantum circuit can be built that fully represents the probabilistic model of the tree. By performing repeated measurements, this Q-tree can then be used for a probabilistic traversal of the tree in order to sample from the probability distribution of a randomly reached leaf. This approach enables a truly random representation of the tree [19, 20]. The results from such a random sampling can be used for a prediction of the class labels of query data. A similar, but slightly modified circuit is also able to provide predictions for uncertain query data. Moreover, a hybrid quantum-classical algorithm can be used for a tree induction by optimizing the corresponding parameters. We propose the usage of a *genetic algorithm* [21] for this purpose.

In short, the conceptional foundation for our approach is a straightforward processing of probability distributions in which we utilize the quantum computer to perform conditional marginalizations based on the parameterized tree structure. Related concepts have already be pursued, e. g., in [22–24] in the context of Bayesian networks.

The remaining paper is structured as follows. We start in Sec. 2 by describing the considered machine learning problem. In Sec. 3, we discuss preliminary considerations regarding purely classical decision trees, which are used in Sec. 4 to present our proposed Q-tree quantum representation. Subsequently, we demonstrate the usage of this quantum representation in Sec. 5 on a quantum computing simulator and actual quantum hardware. We close with conclusions and an outlook in Sec. 6.

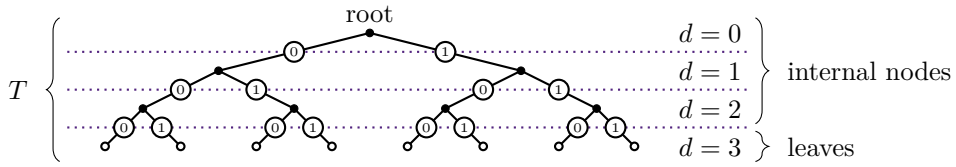


Figure 1: Schematic sketch of a binary classification tree with binary features. Starting from the root, each path along the internal nodes (•) to a leaf (∘) represents a set of decision rules for the data features in the form of Eq. (3) to predict the corresponding labels (i. e., to classify the query data). The outgoing branches represent the two cases $x_i = 0$ (denoted by ⓪) and $x_i = 1$ (denoted by ①), respectively. The depth d indicates the number of decision tests required to reach a node. We only consider trees with no repeating decision rules along a path and no missing internal nodes. The whole tree (of maximum depth 3 in this example) is denoted by T.

2 Problem description

We consider data points of the form $\mathbf{d} \equiv (\mathbf{x}, \mathbf{y})$ with k-dimensional binary input features $\mathbf{x} \in \mathbb{B}^k$ and m-dimensional binary (class) labels $\mathbf{y} \in \mathbb{B}^m$ with $\mathbb{B} \equiv \{0, 1\}$. Based on a given training data set

$$\mathbf{D}_{\text{train}} \equiv \{(\mathbf{x}^1, \mathbf{y}^1), \ldots, (\mathbf{x}^t, \mathbf{y}^t)\} \tag{1}$$

consisting of t data points, a decision tree T can be inducted using an algorithm A such that

$$\mathbf{D}_{\text{train}}, A \mapsto T. \tag{2}$$

The algorithm A can for example represent a global optimization method or a heuristic approach as discussed in Sec. 1.

We limit ourselves to *binary classification trees with binary features* T, which are also known as *binary decision diagrams* [25]. Starting from the root, such a tree consists of a set of nodes connected with branches, where each node has exactly one branch going in. Internal nodes have two branches going out, whereas leaves have no outgoing branches. In each internal node, the test

$$x_i \stackrel{!}{=} 1 \tag{3}$$

is performed for a specific feature $i \in \{1, \ldots, k\}$ and the two outgoing branches represent the rules $x_i = 1$ (i. e., test passed) and $x_i = 0$ (i. e., test failed), respectively. Each node can consequently be associated with a data set representing a subset of the training data that fulfills the decision rules of the path from the root to the respective node.

The *depth* d of a node indicates the number of decision tests required to reach it, where the root corresponds to $d = 0$. For reasons of simplicity, we limit ourselves to trees with the following two properties:

1. A feature cannot be split twice along a path, i. e., decision rules must not repeat.

2. All leaves have the same depth, i. e., the tree is not missing any internal nodes.

The first presumption implies that $d < k$ such that in a tree of depth $k - 1$, all features have to be evaluated once to reach a leaf. At depth d, there are 2^d nodes, each with $k - d$ possible splitting decisions, which in total sum up to $\sum_{l=0}^{d} 2^l (k - l)$ possible decisions along all paths up to this depth. A schematic tree is sketched in Fig. 1.

After its induction, a decision tree T of this form can subsequently be used to predict the labels $\mathbf{y}$ for a query vector of input features $\mathbf{x}$ (i. e., it can be used to classify the query data), which is in general not part of the training data set. However, instead of a crisp prediction value, each leaf of the tree T is labeled with the probabilities of predicting either the label $y_i = 0$ or the label $y_i = 1$ for all $i \in \{1, \ldots, m\}$ as given by the corresponding ratios of labels in the associated subset of the training data. In this sense, we consider *probability estimation trees* that can also be used to

make fuzzy predictions [26]. Specifically, the tree T can predict the label probability distribution $p(\mathbf{y}) \in [0,1]$ from features $\mathbf{x}$ according to

$$T, \mathbf{x} \mapsto p(\mathbf{y}), \tag{4}$$

which also allows to predict the most probable labels $\arg\max_{\mathbf{y}} p(\mathbf{y}) \in \mathbb{B}^m$.

It can be assumed that all data points from the training data set and all query data points are sampled from an underlying "true" (but usually unknown) probability distribution in the sense that

$$\mathbf{d} \sim \tilde{p}(x_1, \ldots, x_k, y_1, \ldots, y_m) \equiv \tilde{p}(\mathbf{x}, \mathbf{y}) \tag{5}$$

for each data point. The induction of a decision tree therefore corresponds to finding a representative estimation for this probability distribution based on the information contained in the training data set.

3 Classical representation

In this section, preliminary considerations regarding purely classical decision trees are presented. First, we describe how a decision tree can be parameterized by a tuple of vectors. This parameterization can the be used for the tree induction. Subsequently, we discuss the probabilistic view of decision trees, in which a marginalized conditional probability distribution can be assigned to each node based on the training data. This formulation can be directly realized with a quantum circuit and is therefore of central importance for our proposed Q-tree approach. Finally, we briefly explain how the prediction of query data fits into the probabilistic framework.

3.1 Tree parameterization

A binary decision tree, as shown in Fig. 1, has a highly symmetric structure, uniquely defined by the feature indices for which a splitting decision is performed in each internal node. We therefore present two possible tree parameterizations in the following, which are particularly useful for the subsequent considerations.

3.1.1 Decision configuration

The structure of a tree T can be parameterized as a tuple of vectors defining the tests in each node. For this purpose, we introduce the *decision configuration*

$$E \equiv (\mathbf{e}^0, \ldots, \mathbf{e}^d) \tag{6}$$

for a tree of maximum depth d as a (d)-tuple of layer (i. e., depth-layer) decisions

$$\mathbf{e}^i \equiv (e^i_1, \ldots, e^i_{2^i}) \in \{1, \ldots, k\}^{2^i} \; \forall \, i \in \{0, \ldots, d\}. \tag{7}$$

The vectors $\mathbf{e^i}$ represents all decisions in depth i, which are defined by the feature indices e^i_j for which a split is made, one for each each node $j \in \{1, \ldots, 2^i\}$ in this depth, sorted in a specific order.

As a convenient sorting order, we propose that given a bit string $\bar{x}_0 \cdots \bar{x}_{l-1}$ of split decisions from the root (i. e., the single node with depth 0) to a node of depth $l \geq 1$ corresponding to the decision rules $x_{e^0} = \bar{x}_0 \wedge \cdots \wedge x_{e^{l-1}} = \bar{x}_{l-1}$, the decision in this node is determined by e^l_ν with

$$\nu \equiv \nu_l(\bar{x}_0 \cdots \bar{x}_{l-1}) \equiv \sum_{j=0}^{l-1} 2^j \bar{x}_{l-1-j} + 1 \tag{8}$$

so that $\nu \in \{1, \ldots, 2^l\}$. Here we assume that a valid sequence of decisions

$$e^i \in \{e^i_1, \ldots, e^i_{2^i}\} \, \forall \, i \in \{0, \ldots, l-1\} \tag{9}$$

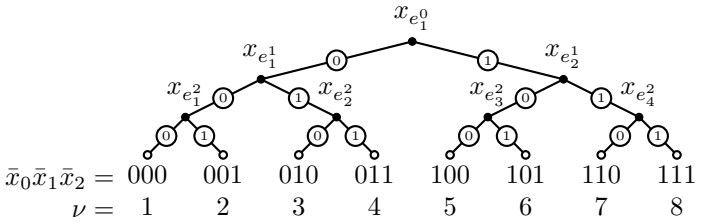


Figure 2: Tree parameterization E, Eq. (6), for the tree from Fig. 1. For each internal node, we show the features for which the respective splitting test, Eq. (3), is performed. In addition, we also show the bit strings of split decisions $\bar{x}_0\bar{x}_1\bar{x}_2$ for the leaves and the corresponding node index ν, Eq. (8). Any node in the tree can be uniquely identified by such a bit string corresponding to the decision rules along the path from the root to this node.

has been chosen (in the sense that this path is actually present in the tree T). For the root, we define $\nu_0 \equiv 1$. Thus, any node in the tree T can be identified by its depth $l \in \{0, \dots, d\}$ and an index ν.

In a valid tree, all feature indices e_j^i must be chosen so that no repeated splits occur along any path. For a tree of depth d, this means that the condition

$$e_1^0 \neq e_j^d \wedge e^l_{\nu_l(\bar{x}_0 \cdots \bar{x}_{l-1})} \neq e_j^d \, \forall \, j \in \{1, \dots, 2^d\} \, \forall \, l \in \{1, \dots, d-1\} \, \forall \, \bar{x}_0, \dots, \bar{x}_{d-1} \in \mathbb{B} \tag{10}$$

has to be fulfilled by the corresponding decision configuration E.

Summarized, the decision configuration E uniquely defines the structure of the tree T as an ordered nested sequence of $\sum_{l=0}^{d} 2^l$ feature indices in total. In addition to its structural information, T also contains information about the training data $\mathbf{D}_{\text{train}}$, Eq. (1), since each node can be associated with the subset of training data that fulfills the corresponding decision rules, which is particularly used in the leaves for the inference of the labels of query data. Combining E and $\mathbf{D}_{\text{train}}$, a one-on-one correspondence

$$T \leftrightarrow (E, \mathbf{D}_{\text{train}}) \tag{11}$$

can be established. An example is sketched in Fig. 2.

3.1.2 Compressed decision configuration

In each layer of the tree, the sequence of decisions, Eq. (8), grows by one feature, which is equivalent to reducing the available number of split choices by one. However, this is not directly reflected by the encoding E, Eq. (6), in which the feature indices e_j^i can attain k possible values independent of the depth. We fix this defect by imposing an additional condition given by Eq. (10). It states that only such values of e_j^i are allowed that produce a tree without repeated splits. Consequently, additional dependencies between the elements of E emerge.

In the context of this work, it turns out to be more practical to switch to a dependency-free parameterization, which naturally reflects the reduction of choices by having less degrees of freedom in each layer. For this purpose, we introduce the *compressed decision configuration*

$$C \equiv (\mathbf{c}^0, \dots, \mathbf{c}^d) \tag{12}$$

for a tree of maximum depth d as a (d)-duple of independent vectors

$$\mathbf{c}^i \equiv (c_1^i, \dots, c_{2^i}^i) \in \{i+1, \dots, k\}^{2^i} \, \forall \, i \in \{0, \dots, d\} \tag{13}$$

in analogy to Eqs. (6) and (7). The elements c_j^i (with $j \in \{1, \dots, 2^i\}$) are mutually independent and correspond to a valid tree without the need for additional constraints. There exists a bijective transformation

$$E \leftrightarrow C \tag{14}$$

between Eqs. (6) and (12) as we show in App. A such that decision tree T can be parameterized by either E, Eqs. (6) and (10), or C, Eq. (12). We consequently use E and C interchangeably.

3.2 Tree induction

The relations in Eqs. (11) and (14) allow us to rephrase Eq. (2) as

$$\mathbf{D}_{\text{train}}, A \mapsto C \tag{15}$$

in the sense that the induction of a tree T based on the training data $\mathbf{D}_{\text{train}}$, Eq. (1), and an algorithm A can equivalently be considered as an induction of the corresponding compressed decision configuration C, Eq. (12). As an example for A, which is particularly useful for our quantum decision tree approach, we propose a genetic algorithm [21, 27], which we briefly discuss in the following.

In general, a genetic algorithm is a gradient-free optimization strategy inspired by natural selection in which a population of candidate solutions is iteratively modified to find the best solution (among the set of evaluated candidates) that maximizes a given fitness function. Typically, a genetic algorithm is defined by three components:

1. A chromosome representation of the solution domain in the sense that each candidate can be uniquely identified by their chromosome.

2. A set of genetic operators to alter the population of candidate solutions (typically in the form of selection, crossover, and mutation).

3. A fitness function, which can be evaluated for each candidate solution.

The resulting solution is only the best solution from the set of evaluated candidates and it cannot be generally guaranteed that it is also globally optimal.

In our case, the goal of the genetic algorithm is to find a decision tree T of a predefined maximal depth d as its solution. Therefore, a possible chromosome representation of a candidate solution is given by the vector

$$\mathbf{C} \equiv (c_1^0, c_1^1, c_2^1, c_1^2, \ldots, c_{2^d}^d) \in \mathcal{S} \tag{16}$$

consisting of the swap indices c_j^i of the corresponding compressed decision configuration C, Eq. (12). The elements of $\mathbf{C}$, which are also called chromosome attributes, are mutually independent and constrained by lower and upper bounds as defined in Eq. (13). These limits define the solution domain

$$\mathcal{S} \equiv \{1, \ldots, k\}^1 \times \cdots \times \{d+1, \ldots, k\}^{2^d} \subseteq \{1, \ldots, k\}^{\sum_{l=0}^{d} 2^l} \tag{17}$$

in such a way that $\mathbf{C}$ can represent any possible tree of maximal depth d. Furthermore, the fitness function can be any map of the form

$$F(\mathbf{C}, \mathbf{D}_{\text{train}}) \mapsto \mathbb{R} \tag{18}$$

and should be chosen in such a way that ordering solutions by their "quality" corresponds to ordering them by their fitness.

A genetic algorithm with these ingredients can be used to find the approximation of the solution

$$\hat{\mathbf{C}}_\theta \approx \arg\max_{\mathbf{C}} F(\mathbf{C}, \mathbf{D}_{\text{train}}) \tag{19}$$

depending on its hyperparameters θ. In App. B, we propose a concrete realization based on this conceptional framework.

3.3 Probabilistic view

Since the training data $\mathbf{D}_{\text{train}}$, Eq. (1), is sampled from a probability distribution $\tilde{p}(\mathbf{x}, \mathbf{y})$, Eq. (5), we can conversely use the probability distribution

$$p(\mathbf{x}, \mathbf{y}) \equiv \frac{|\{\mathbf{d} | (\mathbf{x}', \mathbf{y}') = \mathbf{d} \in \mathbf{D}_{\text{train}} \wedge \mathbf{x}' = \mathbf{x} \wedge \mathbf{y}' = \mathbf{y}\}|}{|\mathbf{D}_{\text{train}}|} \tag{20}$$

based on a histogram of the samples to estimate

$$p(\mathbf{x}, \mathbf{y}) \approx \tilde{p}(\mathbf{x}, \mathbf{y}), \tag{21}$$

where $|\cdot|$ denotes the set cardinality. This estimated probability distribution can be used to establish a probabilistic view of a decision tree by switching from assigning subsets of the data to each node to assigning the corresponding histogram-based probability distributions. However, instead of recounting the histogram for each node, we follow a mathematically equivalent path by successively applying Bayes' rule to construct conditional probability distributions according to the decision rules, as described in the following. A related discussion regarding decision trees and probability distributions can also be found in [28].

Given a tree T, we can obtain the corresponding training data $\mathbf{D}_{\text{train}}$ and decision configuration E according to Eq. (11). The root (where no previously applied decision rules are present) can be directly associated with $p(\mathbf{x}, \mathbf{y})$. The corresponding label probability distribution

$$p(\mathbf{y}) = \sum_{\mathbf{x}} p(\mathbf{x}, \mathbf{y}) \tag{22}$$

can be found by marginalizing over all features $\mathbf{x}$. Here and in the following, a sum over binary variables iterates over $\mathbb{B}$ for each variable.

To obtain the probability distributions associated with deeper nodes, the splitting decisions have to be taken into account. For example, the first split of the tree with respect to the feature $e_1^0 \in \{1, \ldots, k\}$ corresponds to a conditioning on $x_{e_1^0}$ so that the probability distributions assigned to the two branched nodes read

$$p(x_1, \ldots, x_{e_1^0-1}, x_{e_1^0+1}, \ldots, x_k, \mathbf{y} \,|\, x_{e_1^0} = 0) = \frac{p(x_1, \ldots, x_{e_1^0-1}, x_{e_1^0} = 0, x_{e_1^0+1}, \ldots, x_k, \mathbf{y})}{p(x_{e_1^0} = 0)} \tag{23}$$

and

$$p(x_1, \ldots, x_{e_1^0-1}, x_{e_1^0+1}, \ldots, x_k, \mathbf{y} \,|\, x_{e_1^0} = 1) = \frac{p(x_1, \ldots, x_{e_1^0-1}, x_{e_1^0} = 1, x_{e_1^0+1}, \ldots, x_k, \mathbf{y})}{p(x_{e_1^0} = 1)}, \tag{24}$$

respectively. In analogy to Eq. (22), the corresponding label probability distributions are given by

$$p(\mathbf{y} \,|\, x_{e_1^0} = 0) = \sum_{\mathbf{x}_{\setminus\{e_1^0\}}} p(\mathbf{x}_{\setminus\{e_1^0\}}, \mathbf{y} \,|\, x_{e_1^0} = 0) \tag{25}$$

and

$$p(\mathbf{y} \,|\, x_{e_1^0} = 1) = \sum_{\mathbf{x}_{\setminus\{e_1^0\}}} p(\mathbf{x}_{\setminus\{e_1^0\}}, \mathbf{y} \,|\, x_{e_1^0} = 1), \tag{26}$$

respectively, where we have made use of the abbreviation

$$\mathbf{x}_{\setminus S} \equiv \{x_i \,|\, i \in \{1, \ldots, k\} \setminus S\} \subseteq \{x_1, \ldots, x_k\} \tag{27}$$

for a set of indices $S \subseteq \{1, \ldots, k\}$. Here and in the following, sets as arguments of a probability distribution are to be understood in such a way that the corresponding elements are used as arguments to this probability distribution in an appropriate order.

Any node of depth l in the tree T can be identified by its index ν, Eq. (8). Instead of using the index, the node can also be identified by a set of conditions

$$\mathcal{C}_\nu \equiv \mathcal{C}_{\nu_l(\bar{x}_0, \ldots, \bar{x}_{l-1})} \equiv \{x_{e^0} = \bar{x}_{e^0}, \ldots, x_{e^{l-1}} = \bar{x}_{e^{l-1}}\} \tag{28}$$

with feature values $\bar{x}_i \in \mathbb{B}$ for all $i \in \{0, \ldots, l-1\}$ by following the corresponding sequence of decisions $\{e^0, \ldots, e^{l-1}\}$, Eq. (9).

Consequently, any node can be associated with a marginalized conditional probability distribution

$$p(\mathbf{x}_{\setminus\{e^0, \ldots, e^{l-1}\}}, \mathbf{y} \,|\, \mathcal{C}_\nu) = \frac{p(\mathbf{x}_{\setminus\{e^0, \ldots, e^{l-1}\}}, \mathcal{C}_\nu, \mathbf{y})}{p(\mathcal{C}_\nu)} \tag{29}$$

with the probability distribution of the decision set

$$p(\mathcal{C}_\nu) = \sum_{\mathbf{x}_{\setminus\{e^0,\ldots,e^{l-1}\}},\mathbf{y}} p(\mathbf{x}_{\setminus\{e^0,\ldots,e^{l-1}\}}, \mathcal{C}_\nu, \mathbf{y}) \tag{30}$$

and the corresponding label probability distribution

$$p(\mathbf{y}\,|\,\mathcal{C}_\nu) = \sum_{\mathbf{x}_{\setminus\{e^0,\ldots,e^{l-1}\}}} p(\mathbf{x}_{\setminus\{e^0,\ldots,e^{l-1}\}}, \mathbf{y}\,|\,\mathcal{C}_\nu), \tag{31}$$

which can also be marginalized to get the probability distribution

$$p(y_i\,|\,\mathcal{C}_\nu) = \sum_{y_1,\ldots,y_{i-1},y_{i+1},\ldots,y_m} p(\mathbf{y}\,|\,\mathcal{C}_\nu) \tag{32}$$

of the ith label with $i \in \{1,\ldots,m\}$. Thus, we have established a probabilistic relation between the training data $\mathbf{D}_{\text{train}}$ and the tree structure E, Eq. (6).

A probabilistic tree traversal corresponds to drawing a sample $\bar{\mathbf{x}} \in \mathbb{B}^k$ from

$$p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x}, \mathbf{y}), \tag{33}$$

which is based on Eq. (21), and subsequently using the decisions $\mathcal{C}_\nu$ in the tree T to reach a certain node. The probability distribution of reaching this node is then given by $p(\mathcal{C}_\nu)$ and the corresponding label probability by $p(\mathbf{y}\,|\,\mathcal{C}_\nu)$. Equivalently, the probabilistic tree traversal can also be understood as drawing a sample $\bar{\mathbf{x}} \in \mathbb{B}^k$ and $\bar{\mathbf{y}} \in \mathbb{B}^m$ directly from

$$p(\mathcal{C}_\nu, \mathbf{y} = \bar{\mathbf{y}}) = \sum_{\mathbf{x}_{\setminus\{e^0,\ldots,e^{l-1}\}}} p(\mathbf{x}_{\setminus\{e^0,\ldots,e^{l-1}\}}, \mathcal{C}_\nu, \mathbf{y})|_{\mathbf{y}=\bar{\mathbf{y}}}, \tag{34}$$

which allows to infer $p(\mathcal{C}_\nu)$ and $p(\mathbf{y} = \bar{\mathbf{y}}\,|\,\mathcal{C}_\nu)$, respectively.

3.4 Tree predictions

Given a tree T of maximal depth d, the prediction of query data $\mathbf{x}^q \in \mathbb{B}^k$ in the sense of Eq. (4) can be performed by an iterative traversal of the tree. Starting from the root, the index of the traversed node of depth 1 (after the first split with feature index e_1^0) is given by

$$q_1 \equiv q_1(\mathbf{x}^q) = \nu_1(x^q_{e_1^0}), \tag{35}$$

the index of the subsequently traversed node of depth 2 (after the second split with feature index $e^1_{q_1}$) by

$$q_2 \equiv q_2(\mathbf{x}^q) = \nu_2(x^q_{e_1^0} x^q_{e^1_{q_1}}) \tag{36}$$

and so on, where we recall Eq. (8). Generally, the the index of the traversed node of depth $l > 1$ reads

$$q_l \equiv q_l(\mathbf{x}^q) = \nu_l(x^q_{e_1^0} x^q_{e^1_{q_1}} \cdots x^q_{e^{l-1}_{q_{l-1}}}) \tag{37}$$

until for $l = d$ the leaf with index q_d is reached.

The traversed path imposes the set of conditions

$$\mathcal{C}_{q_d(\mathbf{x}^q)} = \{x_{e_1^0} = x^q_{e_1^0}, \ldots, x_{e^{d-1}_{q_{d-1}}} = x^q_{e^{d-1}_{q_{d-1}}}\} \tag{38}$$

according to Eq. (28). Consequently, the predicted probability distribution for ith label with $i \in \{1,\ldots,m\}$ reads

$$\hat{p}(y_i\,|\,\mathbf{x}^q) \equiv p(y_i\,|\,\mathcal{C}_{q_d(\mathbf{x}^q)}), \tag{39}$$

where we recall Eq. (32). This expression also allows to infer the most probable label as

$$\hat{y}_i \equiv \arg\max_{y_i} p(y_i\,|\,\mathcal{C}_{q_d(\mathbf{x}^q)}), \tag{40}$$

where we recall Eq. (32).

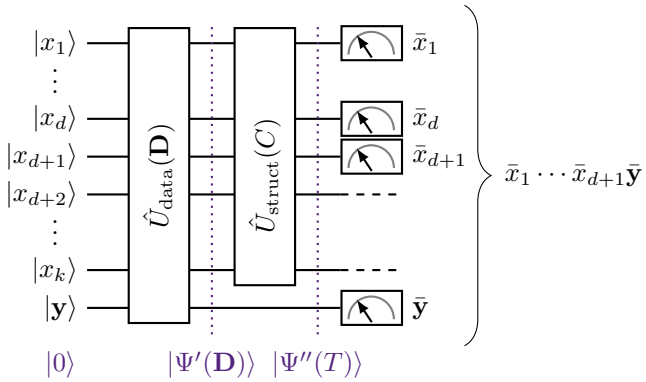


Figure 3: Q-tree (circuit). Layout for our proposed quantum representation of the classical tree T of maximum depth d, which is uniquely defined by the training data $\mathbf{D}_{\text{train}}$ and the compressed decision configuration C, Eq. (12). The circuit contains $k+m$ qubits in total, where the first k qubits $|x_1\rangle, \ldots, |x_k\rangle$ represent the features and the next m qubits $|y_1\rangle, \ldots, |y_m\rangle$ represent the labels. Initially, all qubits are prepared in the ground state $|0\rangle$, Eq. (41). The first unitary operator $\hat{U}_{\text{data}}(\mathbf{D}_{\text{train}})$ performs an encoding of the training data that leads to the state $|\Psi'(\mathbf{D}_{\text{train}})\rangle$, Eq. (42). The second unitary operator $\hat{U}_{\text{struct}}(C)$ performs an encoding of the decision configuration that leads to the state $|\Psi''(T)\rangle$, Eq. (43). Third, a projective measurement on the first $d+1$ qubits and the last m qubits is performed, Eq. (56). Since the measurement results are drawn from a probability distribution that is associated with a randomly selected leaf of T, Eq. (56), the circuit can be considered a quantum representation of this tree.

4 Quantum representation

So far, our considerations have been completely classical. In this section, we propose Q-trees as a quantum representation of binary classification trees with binary features using quantum circuits [29]. Our considerations are mainly based on the probabilistic perspective on decision trees from Sec. 3.3.

Specifically, we presume that there is a tree T of maximum depth d, which is uniquely defined by the training data $\mathbf{D}_{\text{train}}$, Eq. (1), and the compressed decision configuration C, Eq. (12). Any leaf of T can be associated with a probability distribution, Eq. (29), that describes the probability distribution of $\mathbf{D}_{\text{train}}$, Eq. (21), conditioned by the decisions along the path from the root to the leaf. Based on this perspective, we can construct a quantum circuit that performs a random traversal of the tree in the sense that it samples the label probability $p(\mathbf{y} \,|\, \mathcal{C}_\nu)$, Eq. (31), of a random leaf that is selected with probability $p(\mathcal{C}_\nu)$, where $\mathcal{C}_\nu$, Eq. (28), denotes the set of decisions that identifies a path in the tree. We achieve this by preparing the probability distribution of $p(\mathbf{x}, \mathbf{y})$, Eq. (21), in terms of qubit amplitudes and subsequently apply a set of (conditional) SWAP gates to alter the amplitudes according to the tree structure. The final projective measurement over a subset of the qubits then yields a sample from $p(\mathcal{C}_\nu, \mathbf{y} = \bar{\mathbf{y}})$, Eq. (34).

In the following, we explain this approach in more detail. We start by describing the corresponding quantum circuit and how it can be constructed from a given tree. In the subsequent part, we briefly discuss how tree inductions and label predictions of query data can be performed within this framework.

4.1 Tree circuit

A Q-tree representing the classical tree T can be realized by using the circuit layout sketched in Fig. 3. This *Q-tree circuit*, which we also refer to as Q-tree for short, contains $k+m$ qubits in total. The first k qubits (denoted by $|x_1\rangle, \ldots, |x_k\rangle$ or their respective indices $1, \ldots, k$) represent the features, whereas the next m qubits (denoted by $|y_1\rangle, \ldots, |y_m\rangle$ or their respective indices $k+1, \ldots, k+m$) represents the labels.

Initially, all qubits are assumed to be prepared in the ground state

$$|\Psi\rangle \equiv |0\rangle \,. \tag{41}$$

In the next step, two unitary operators are applied. The first is responsible for the encoding of the training data and the second for the encoding of the structure (i. e., the decision configuration) of the tree. Finally, a projective measurement on the first $d+1$ qubits and the last m qubits is performed. We discuss the components of the circuit in more detail below.

4.1.1 Data encoding

The first unitary operator $\hat{U}_{\text{data}}(\mathbf{D}_{\text{train}})$ in the Q-tree, Fig. 3, is responsible for an encoding of the training data $\mathbf{D}_{\text{train}}$, Eq. (1), in form of quantum amplitudes [30]. By definition, it acts on all $k+m$ qubits and leads to the state

$$|\Psi'(\mathbf{D}_{\text{train}})\rangle \equiv \hat{U}_{\text{data}}(\mathbf{D}_{\text{train}})\,|0\rangle = \sum_{\mathbf{x},\mathbf{y}} \sqrt{p(\mathbf{x},\mathbf{y})}\,|\mathbf{x},\mathbf{y}\rangle\,, \tag{42}$$

where we recall $p(\mathbf{x},\mathbf{y})$, Eq. (20). The encoded state $|\Psi'(\mathbf{D}_{\text{train}})\rangle$ contains all $k+m$ (possibly entangled) qubits. This data encoding, where features are encoded in qubits and their respective probabilities in the state amplitudes such that the amplitude vector represents a classical discrete probability distribution, is also referred to as *qsample encoding* [22, 31, 32].

Classically, storing the total probability distribution requires $\mathcal{O}(2^{k+m})$ classical memory slots. Since only $k+m$ qubits are required to store the complete information from $p(\mathbf{x},\mathbf{y})$, the qsample encoding achieves an exponential compression. To obtain $\hat{U}_{\text{data}}(\mathbf{D}_{\text{train}})$ for a given $\mathbf{D}_{\text{train}}$, classical algorithms can be used [33–35]. Recent methods include *black-box state preparation without arithmetics* [36, 37] and *quantum generative adversarial networks* [38–41]. We do not further discuss these approaches here and instead assume that an appropriate data encoding operator is available to realize Eq. (42). In general, $\hat{U}_{\text{data}}(\mathbf{D}_{\text{train}})$ consists of $\mathcal{O}(2^{k+m})$ gate operations [42] but can be simplified if the probability distribution is not fully correlated [22]. The complexity can be further reduced if ancilla qubits are used [43–45], an approach which we do not further pursue in this manuscript.

4.1.2 Structure encoding

It is well-known that quantum circuits with a qsample encoding can be used to manipulate probability distributions, for example to perform marginalizations [22, 32]. This circumstance can be exploited to construct a unitary operator with a conditional mapping of the initial probability distribution of the data to marginalized conditional probability distributions in individual leaves of the tree. This is the conceptional idea behind the structure encoding (or splitting decision encoding), which we explain in more detail in the following.

The second unitary operator $\hat{U}_{\text{struct}}(C)$ in the Q-tree, Fig. 3, is responsible for this structure encoding according to

$$|\Psi''(T)\rangle \equiv \hat{U}_{\text{struct}}(C)\,|\Psi'(\mathbf{D}_{\text{train}})\rangle\,. \tag{43}$$

It only acts on the first k qubits and is determined by the compressed decision configuration C, Eq. (12). Specifically, each layer swap $\mathbf{c}^i$ with $i \in \{0,\dots,d\}$, Eq. (13), can be associated with 2^i (multi-controlled) SWAP gates, which are partially "decorated" by NOT gates. The NOT gates are added in such a way that each combination of decorated and non-decorated controlled qubits is placed once, each assigned to one element c^i_j of $\mathbf{c}^i$ with $j \in \{1,\dots,2^i\}$.

This sequence of gates can be expressed by

$$\hat{U}_{\text{struct}}(C) = \prod_{i=0}^{d} \hat{U}^i_{\text{struct}}(\mathbf{c}^i) \tag{44}$$

with

$$\hat{U}^i_{\text{struct}}(\mathbf{c}^i) \equiv \prod_{j=1}^{2^i} \hat{\Gamma}^i_j(c^i_j) \tag{45}$$

for each layer swap. The products in Eqs. (44) and (45) are ordered products in the sense of

$$\prod_{i=a}^{b} \hat{O}_i \equiv \hat{O}_b \hat{O}_{b+1} \cdots \hat{O}_{a-1} \hat{O}_a \tag{46}$$

for a set of unitary operators $\hat{O}_i$ for $i \in \{a, \ldots, b\}$ and $a \leq b$. Furthermore, we introduce

$$\hat{\Gamma}^i_j(c) \equiv \hat{\mu}^i_j \, \hat{\gamma}^i(c) \, \hat{\mu}^i_j, \tag{47}$$

with $c \in \{i+1, \ldots, k\}$. It contains the abbreviations

$$\hat{\mu}^i_j \equiv \begin{cases} \hat{\eta}^i_j & \text{for } i > 0 \\ \mathbb{1} & \text{for } i = 0 \end{cases} \tag{48}$$

and

$$\hat{\gamma}^i(c) \equiv \begin{cases} \mathrm{C}_i\mathrm{SWAP}(i+1, c, \{1, \ldots, i\}) & \text{for } i \geq 1 \\ \mathrm{SWAP}(i+1, c) & \text{for } i = 0 \end{cases} \tag{49}$$

based on

$$\hat{\eta}^v_j \equiv \prod_{u=1}^{v} \begin{cases} \mathrm{NOT}(u) & \text{for } \kappa(j, v, u) = 0 \\ \mathbb{1} & \text{otherwise} \end{cases} \tag{50}$$

with $v \in \{1, \ldots, d\}$. The latter also makes use of

$$\kappa(j, v, u) \equiv \left\lfloor \frac{j-1}{2^{v-u}} \right\rfloor \bmod 2 \in \mathbb{B} \tag{51}$$

for $u \in \{1, \ldots, v\}$.

The unitary operators $\hat{\mu}^i_j$ and $\hat{\gamma}^i(c)$ represent one of four standard gates [29]:

- An identity gate, denoted by $\mathbb{1}$, which has no effect on the state.
- A NOT or Pauli X gate acting on the qth qubit with $q \in \{1, \ldots, k\}$, which is denoted by $\mathrm{NOT}(q)$.
- A SWAP gate with respect to two target qubits q_1 and q_2 (with $q_1, q_2 \in \{1, \ldots, k\}$), which is denoted by $\mathrm{SWAP}(q_1, q_2)$. For identical swapping qubits $q_1 = q_2$, one has $\mathrm{SWAP}(q_1, q_1) = \mathbb{1}$.
- A (multi-)controlled SWAP gate with respect to two target qubits q_1 and q_2 to swap (with $q_1, q_2 \in \{1, \ldots, k\}$) and $v \geq 1$ control qubits defined by the non-empty set $Q \in \{1, \ldots, k\}^v$, which is denoted by $\mathrm{C}_v\mathrm{SWAP}(q_1, q_2, Q)$ and acts on $v+2$ qubits. $\mathrm{C}_1\mathrm{SWAP}$ represents a CSWAP or Fredkin gate. Furthermore, for identical swapping qubits $q_1 = q_2$, one has $\mathrm{C}_v\mathrm{SWAP}(q_1, q_1, Q) = \mathbb{1}$. To simplify our notation, we also write $\mathrm{C}_0\mathrm{SWAP}(q_1, q_2, \{\}) \equiv \mathrm{SWAP}(q_1, q_2)$.

The operations in $\hat{U}_{\mathrm{struct}}(C)$ can consequently be considered as a sequence of these standard gates. The NOT, SWAP, and (multi-)controlled SWAP gates are understood to act as identity gates (i. e., $\mathbb{1}$) on the remaining qubits.

An example circuit is outlined in Fig. 4. It shows the connection between the structure of a decision tree T (which is parameterized by the decision configuration E, Eq. (6), and the compressed decision configuration C, Eq. (12), with the correspondence from Eq. (14)) and its Q-tree encoding via Eq. (44). The quantum representation of the tree structure of all nodes of depth d requires up to $d2^d$ NOT gates and up to 2^d $\mathrm{C}_d\mathrm{SWAP}$ gates. However, at least $\sum_{l=1}^{d-1} 2^d(1 - 2^{-l})$ of the NOT gates are consecutively applied to the same qubit and therefore eliminate each other.

To estimate the complexity of the structure encoding circuit, we decompose it into one- and two-qubit gate operations [42]. For this purpose, we make use of the relation [46]

$$\mathrm{C}_v\mathrm{SWAP}(q_1, q_2, Q) = \mathrm{CNOT}(q_2, q_1) \; \mathrm{C}_{v+1}\mathrm{NOT}(q_1, Q \cup \{q_1\}) \; \mathrm{CNOT}(q_2, q_1) \tag{52}$$

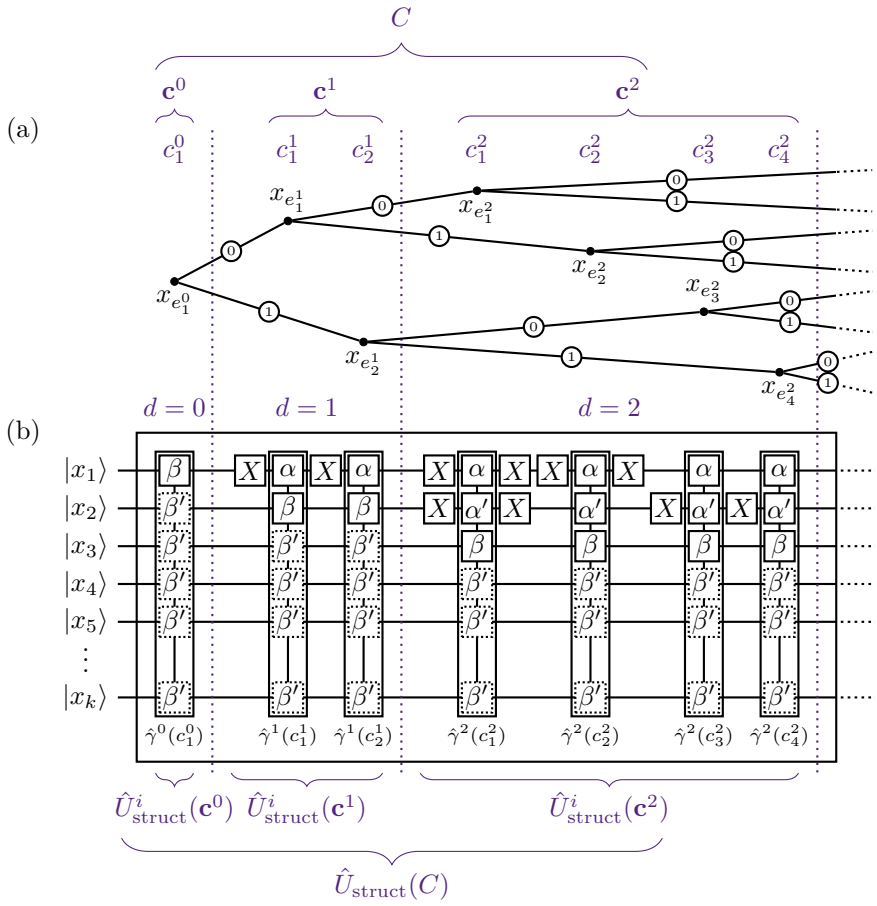


Figure 4: Correspondence between a decision tree T and the Q-tree encoding of its structure given by $\hat{U}_{\text{struct}}(C)$, Eq. (44), which is the second unitary operator in the Q-tree, Fig. 3. (a) Decision tree from Fig. 2 with splitting feature indices e_j^i as given by the decision configuration E, Eq. (6), and corresponding swap indices c_j^i as given by the compressed decision configuration C, Eq. (12), where Eq. (14) holds true. (b) Structural encoding transformation consisting of a sequence of layer transformations $\hat{U}^i_{\text{struct}}(\mathbf{c}^i)$, Eq. (45), which in turn are sequences of NOT and $\hat{\gamma}^i(c)$ gates. The latter represent a SWAP gate or a C_iSWAP gate as defined in Eq. (49). We denote NOT gates with X and show the individual qubits affected by the SWAP and C_iSWAP gates: control qubits are marked by α and α', respectively, whereas qubits to swap and their potential swapping partners are marked by β and β', respectively. Thus, one has $\hat{\gamma}^0(c_j^0) = \mathrm{SWAP}(\beta, \beta')$ for $d = 0$, $\hat{\gamma}^1(c_j^1) = \mathrm{C_1SWAP}(\beta, \beta', \{\alpha\})$ for $d = 1$, and $\hat{\gamma}^2(c_j^2) = \mathrm{C_2SWAP}(\beta, \beta', \{\alpha, \alpha'\})$ for $d = 2$. The parameters α, α', and β are determined by d, whereas β' is determined by the corresponding swap index c_j^i. Directly consecutive NOT gates eliminate each other and can therefore be omitted, but we nevertheless show them for the sake of completeness.

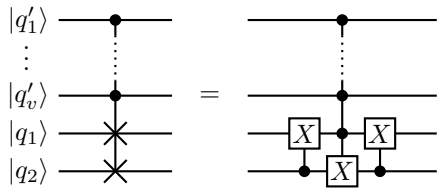


Figure 5: Decomposition of a (multi-controlled) SWAP gate into two CNOT gates and a (multi-)controlled NOT gate as given by Eq. (52).

for $q_1 \neq q_2$ with $v \geq 0$ control qubits, which is also sketched in Fig. 5. Here, $\mathrm{C}_v\mathrm{NOT}(q, Q)$ denotes a (multi-)controlled NOT gate acting on qubit q with $v \geq 1$ control qubits defined by the non-empty set $Q \equiv \{q'_1, \ldots, q'_v\}^v$ with $q, q'_1, \ldots, q'_v \in \{1, \ldots, k\}$. For $v = 1$, $\mathrm{C}_1\mathrm{NOT}(q, \{q'_1\})$ represents a two-qubit CNOT gate, which is also denoted by $\mathrm{CNOT}(q, q'_1)$. For $v > 2$, $\mathrm{C}_v\mathrm{NOT}(q, Q)$ is also called (multi-controlled) Toffoli gate. The $\mathrm{C}_v\mathrm{NOT}$ gate can be further decomposed in various ways [47], depending on whether additional ancilla qubits are to be used or not. We particularly consider an ancilla-free decomposition into $8v - 20$ two-qubit controlled-rotation gates [48]. Thus, the structure encoding of a tree T with depth d requires a total of $\mathcal{O}(d2^d)$ NOT gates and $\mathcal{O}(d)$ two-qubit controlled-rotation gates, resulting in an effective complexity of $\mathcal{O}(d2^d)$ gates.

As visualized in Fig. 4, each unitary operator $\hat{\Gamma}^i_j(c)$, Eq. (47), can be associated with a node of depth i and index j in T. According to the definition of the operator, NOT gates appear only in pairs and therefore effectively only $\mathrm{C}_i\mathrm{SWAP}$ gates are applied. In particular, the NOT decorations are exclusively attached to control qubits before and after a $\mathrm{C}_i\mathrm{SWAP}$ and are therefore responsible for flipping the controls temporarily. The placement of NOT gates is determined by $\kappa(j, v, u)$, Eq. (51), which decides whether the uth qubit acts as an open control gate (for $\kappa(j, v, u) = 0$) or a closed control gate [46]. The application of $\hat{\Gamma}^i_j(c)$ to an arbitrary entangled k-qubit state $\sum_{\mathbf{x}} a(\mathbf{x}) |\mathbf{x}\rangle$ with amplitudes $a(\mathbf{x})$ consequently yields

$$\hat{\Gamma}^i_j(c) \sum_{\mathbf{x}} a(\mathbf{x}) |\mathbf{x}\rangle = \sum_{\mathbf{x}} a_\Gamma(\mathbf{x}) |\mathbf{x}\rangle \tag{53}$$

with

$$a_\Gamma(\mathbf{x}) \equiv \begin{cases} a(x_1, \ldots, x_i, x_c, x_{i+2}, \ldots, x_{c-1}, x_{i+1}, x_{c+1}, \ldots, x_k) & \text{for } x_q = \kappa(j, i, u) \,\forall\, u \in \{1, \ldots, i\} \\ a(\mathbf{x}) & \text{otherwise} \end{cases} \tag{54}$$

for $i \geq 1$ and, without loss of generality, $i+1 < c$. Clearly, the only transformation of the amplitudes is a conditional swap of indices. In case of $i = 0$, the index swap is performed unconditioned and for $i + 1 = c$, one has $a_\Gamma(\mathbf{x}) = a(\mathbf{x})$.

4.1.3 Measurement

The projective measurement at the end of the Q-tree yields a bit string

$$\bar{x}_1 \cdots \bar{x}_{d+1} \bar{\mathbf{y}} \sim p_T(\bar{x}_1, \ldots, \bar{x}_{d+1}, \bar{\mathbf{y}}) \tag{55}$$

with

$$\begin{aligned} p_T(\bar{x}_1, \ldots, \bar{x}_{d+1}, \bar{\mathbf{y}}) &\equiv p_T(x_1 = \bar{x}_1, \ldots, x_{d+1} = \bar{x}_{d+1}, \mathbf{y} = \bar{\mathbf{y}}) \\ &= \mathrm{Tr}\{\langle \bar{x}_1, \ldots, \bar{x}_{d+1}, \bar{\mathbf{y}} | \Psi''(T) \rangle \langle \Psi''(T) | \bar{x}_1, \ldots, \bar{x}_{d+1}, \bar{\mathbf{y}} \rangle\}_{\bar{x}_{d+2}, \ldots, \bar{x}_k} \\ &= \sum_{x_{d+2}, \ldots, x_k} p_T(\mathbf{x}, \bar{\mathbf{y}})|_{x_1 = \bar{x}_1, \ldots, x_{d+1} = \bar{x}_{d+1}} \\ &= \sum_{\mathbf{x}_{\setminus \{e^0, \ldots, e^d\}}} p(\mathbf{x}, \bar{\mathbf{y}})|_{x_{e^0} = \bar{x}_1, \ldots, x_{e^d} = \bar{x}_{d+1}} \\ &= p(x_{e^0} = \bar{x}_1, \ldots, x_{e^d} = \bar{x}_{d+1}, \mathbf{y} = \bar{\mathbf{y}}) = p(\mathcal{C}_\nu, \mathbf{y} = \bar{\mathbf{y}}), \end{aligned} \tag{56}$$

as shown in App. C, where we recall $\mathcal{C}_\nu = \mathcal{C}_{\nu(\bar{x}_1,\ldots,\bar{x}_{d+1})}$, Eq. (28), and $p(\mathcal{C}_\nu, \mathbf{y} = \bar{\mathbf{y}})$, Eq. (34). Thus, we can infer the probability distribution of reaching a certain leaf, Eq. (30), from

$$p_T(\bar{x}_1,\ldots,\bar{x}_{d+1}) = \sum_{\mathbf{y}} p_T(\bar{x}_1,\ldots,\bar{x}_{d+1},\mathbf{y}) = p(\mathcal{C}_\nu), \tag{57}$$

and the label probability distribution, Eq. (31), from

$$p_T(\bar{\mathbf{y}}|\bar{x}_1,\ldots,\bar{x}_{d+1}) = \frac{p_T(\bar{x}_1,\ldots,\bar{x}_{d+1},\mathbf{y})}{p_T(\bar{x}_1,\ldots,\bar{x}_{d+1})} = p(\mathbf{y}|\mathcal{C}_\nu)|_{\mathbf{y}=\bar{\mathbf{y}}}, \tag{58}$$

respectively. In addition, we can also obtain the probability distribution

$$p_T(\bar{y}_i|\bar{x}_1,\ldots,\bar{x}_{d+1}) = \sum_{y_1,\ldots,y_{i-1},y_{i+1},\ldots,y_m} p_T(\mathbf{y}|\bar{x}_1,\ldots,\bar{x}_{d+1})|_{\mathbf{y}=\bar{\mathbf{y}}} = p(y_i|\mathcal{C}_\nu)|_{y_i=\bar{y}_i}, \tag{59}$$

of the ith label with $i \in \{1,\ldots,m\}$, Eq. (32).

Summarized, we have shown that measuring the Q-tree (i. e., the circuit shown in Fig. 3 with $k+m$ qubits and a complexity of $\mathcal{O}(2^{k+m}+d2^d)$ gates) corresponds to drawing a sample from the conditional probability distribution $p(\mathcal{C}_\nu, \mathbf{y} = \bar{\mathbf{y}})$, Eq. (34), that is associated with the randomly selected leaf of depth d and index ν of the respective decision tree T. The probability that a certain leaf is selected is given by the corresponding probability distribution $p(\mathcal{C}_\nu)$, Eq. (30). Consequently, the Q-tree can be considered a quantum representation of T.

Conversely, from multiple measurements we can estimate $p(\mathcal{C}_\nu)$ and $p(y_i|\mathcal{C}_\nu)$, respectively. For this purpose, an ensemble of bit strings $\{b_1,\ldots,b_N\}$ is collected from a sequence of N measurements. For each bit string, the corresponding set of conditions $\mathcal{C}_\nu$, Eq. (28), is extracted from the fist $d+1$ bits and the corresponding labels $\bar{\mathbf{y}}$ from the next m bits. The number of measurements $n(\mathcal{C}_\nu)$ that fulfills the set of conditions $\mathcal{C}_\nu$ is collected and divided into $2m$ bins. First, m bins $n_j^0(\mathcal{C}_\nu)$ with $j \in \{1,\ldots,m\}$, for which the $(d+1+j)$th bit is 0 (i. e., $\bar{y}_j = 0$). And second, m bins $n_j^1(\mathcal{C}_\nu)$, for which the $(d+1+j)$th bit is 1. By definition, one has $\sum_{\mathcal{C}_\nu} n(\mathcal{C}_\nu) = N$ and $n(\mathcal{C}_\nu) = n_j^0(\mathcal{C}_\nu) + n_j^1(\mathcal{C}_\nu)$ for all $j \in \{1,\ldots,m\}$, respectively.

Based on these measurement results, we can obtain the approximations

$$p(\mathcal{C}_\nu) \approx \hat{p}(\mathcal{C}_\nu) \equiv \frac{n(\mathcal{C}_\nu)}{N} \tag{60}$$

and

$$p(y_i\,|\,\mathcal{C}_\nu) \approx \hat{p}(y_i\,|\,\mathcal{C}_\nu) \equiv \frac{n_i^{y_i}(\mathcal{C}_\nu)}{n(\mathcal{C}_\nu)} \text{ if } n(\mathcal{C}_\nu) > 0, \tag{61}$$

respectively. For $n(\mathcal{C}_\nu) = 0$ (i. e., none of the measured bit strings fulfills the set of conditions $\mathcal{C}_\nu$), Eq. (61) is undefined. The (statistical) uncertainties of Eqs. (60) and (61) are discussed in App. D.

The sampling from measurements corresponds to a truly random tree traversal the sense that it relies on intrinsic quantum randomness [49]. If we assume that the training data $\mathbf{D}_{\text{train}}$ is sufficiently large such that Eq. (21) holds true, such samples are in fact approximately drawn from the underlying distribution $\tilde{p}(\mathcal{C}_\nu, \mathbf{y} = \bar{\mathbf{y}})$, Eq. (5). In the following, we briefly explain how we can use the proposed setup to induce trees or to predict the labels of (uncertain) query data.

4.2 Tree induction

To induce a Q-tree in the sense of Eq. (15), we consider it as a parameterized (or variational) circuit [39]. Specifically, the Q-tree is according to Eqs. (42) and (43) parameterized by the training data $\mathbf{D}_{\text{train}}$, Eq. (1), and the compressed decision configuration C, Eq. (12). Since the training data is supposed to be immutable, the variable parameters are represented only by C. Consequently, a variational quantum algorithm can be used to optimized the parameterized circuit according to some optimization goal [50]. Following this strategy, we employ a genetic algorithm in the same way as for the classical tree. For this purpose, we replace the fitness function, Eq. (18), by

$$\hat{F}(b_1,\ldots,b_N,\mathbf{D}_{\text{train}}) \mapsto \mathbb{R}, \tag{62}$$

where $\{b_1, \ldots, b_N\}$ represents an ensemble of N bit strings, Eq. (55), collected from a sequence of N measurements of the Q-tree. Due to the noisy nature of the measurement results from quantum circuits, $\hat{F}(b_1, \ldots, b_N, \mathbf{D}_{\text{train}})$ can in fact be considered a noisy fitness function [51]. A concrete example for Eq. (62) is proposed in App. B.

The use of genetic algorithms in the context of quantum computing is a well-known field [52–54]. Recent results particularly highlight the potential of evolutionary methods for parameterized circuit optimization in the presence of noisy quantum hardware [55–57]. However, there are still unresolved practical challenges such as barren plateaus [58].

4.3 Tree predictions

To predict the labels of query data $\mathbf{x}^Q \in \mathbb{B}^k$, we assume that the estimated label probability distribution $\hat{p}(y_i \,|\, \mathcal{C}_\nu)$, Eq. (61), is known for all $\nu \in \{1, \ldots, 2^d\}$ from a previous sampling. In this case, there are two alternative prediction approaches, which we present in the following.

The first prediction approach corresponds to a classical evaluation of the tree in the sense that the knowledge about the decision configuration E, Eq. (6), allows a conditional traversal of the tree according to the elements of $\mathbf{x}^q$ until a leaf is reached. Thus, the predicted probability distribution for the ith label with $i \in \{1, \ldots, m\}$ reads

$$\hat{p}_{\text{cl}}(y_i \,|\, \mathbf{x}^q) \equiv \hat{p}(y_i \,|\, \mathcal{C}_{q_d(\mathbf{x}^q)}) \tag{63}$$

in analogy to Eq. (39), where we recall $\mathcal{C}_{q_d(\mathbf{x}^q)}$, Eq. (38). Furthermore, we can use

$$\hat{y}_i \equiv \arg\max_{y_i} \hat{p}_{\text{cl}}(y_i \,|\, \mathbf{x}^q) \tag{64}$$

in analogy to Eq. (40) to predict labels.

The second prediction approach corresponds to a semi-quantum evaluation of the tree. This approach allows to process a probability distribution of query data

$$p^q \equiv p^q(\mathbf{x}^q) \tag{65}$$

with support $\mathbb{B}^k$ instead of a single data point, which enables the consideration of uncertainty in the query data. For this purpose, the evaluation of the quantum circuit outlined in Fig. 6 is required, which we refer to as *query circuit*. It contains k qubits, which are assumed to be initially prepared in the ground state $|0\rangle$, Eq. (41), and consists of two consecutive unitary operators followed by a projective measurement of the first $d+1$ qubits. The first unitary operator $\hat{U}_{\text{query}}(p^q)$ is responsible for a qsample encoding of $p^q(\mathbf{x}^q)$ in analogy to $\hat{U}_{\text{data}}(\mathbf{D}_{\text{train}})$, Eq. (42), whereas the second unitary operator corresponds to $\hat{U}_{\text{struct}}(C)$, Eq. (44), and is therefore responsible for the structural encoding of the tree T. In particular, p^q can also represent a single query data point $\mathbf{x}^q$ by setting $p^q(\mathbf{x}^q) = 1$. The qsample data encoding can in this case be achieved straightforwardly by applying a suitably chosen array of NOT gates.

A measurement yields the a string

$$\bar{x}_1^q \cdots \bar{x}_d^q \sim p_{(p^q, C)}(\bar{x}_1, \ldots, \bar{x}_{d+1}) \tag{66}$$

with

$$p_{(p^q, C)}(\bar{x}_1, \ldots, \bar{x}_{d+1}) = \text{Tr}\{\langle \bar{x}_1^q, \ldots, \bar{x}_{d+1}^q | \Psi^q(p^q, C)\rangle \langle \Psi^q(p^q, C) | \bar{x}_1^q, \ldots, \bar{x}_{d+1}^q \rangle\}_{\bar{x}_{d+2}, \ldots, \bar{x}_k} \tag{67}$$

and

$$|\Psi^q(p^q, C)\rangle \equiv \hat{U}_{\text{struct}}(C) \hat{U}_{\text{query}}(p^q) \,|0\rangle \,. \tag{68}$$

We obtain an ensemble $\{b_1, \ldots, b_N\}$ of such bit strings from N measurements and count the number of measurements $n(\mathcal{C}_\nu)$ that fulfill a set of conditions $\mathcal{C}_\nu$, Eq. (28), such that $\sum_{\mathcal{C}_\nu} n(\mathcal{C}_\nu) = N$. Thus, we can estimate the probability distribution of reaching a certain leaf

$$\hat{p}^q(\mathcal{C}_\nu \,|\, p^q) \equiv \frac{n(\mathcal{C}_\nu)}{N}, \tag{69}$$

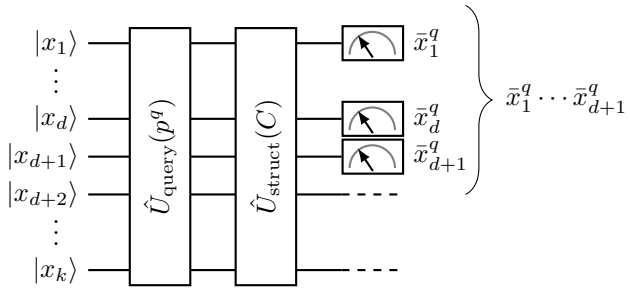


Figure 6: Query circuit. Layout for the prediction of labels for a probability distribution of query data $p^q(\mathbf{x}^q)$, Eq. (65), using a Q-tree, Fig. 3. The circuit contains k qubits and consists of the unitary operator $\hat{U}_{\text{query}}(p^q)$ for the qsample encoding of $p^q(\mathbf{x}^q)$, the unitary operator $\hat{U}_{\text{struct}}(C)$, Eq. (44), for the structural encoding of the tree T and a subsequent projective measurement of the first $d+1$ qubits. The measurement result, Eq. (67), can be used for the prediction of labels of uncertain query data, Eq. (70).

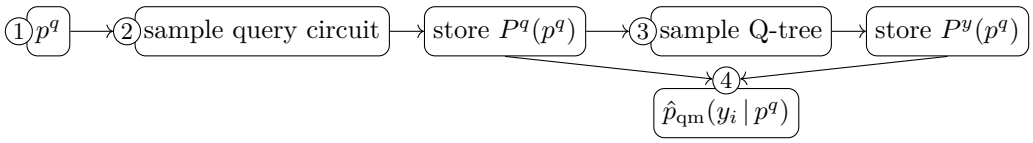


Figure 7: Sketch of the proposed on-demand sampling method to perform Q-tree predictions with a constant number of classical memory slots, which consists of four steps. First (①), the probability distribution of query data $p^q(\mathbf{x}^q)$, Eq. (65), is chosen. Second (②), the query circuit from Fig. 6 is measured N times to sample $\hat{p}^q(\mathcal{C}_\nu)$, Eq. (69). The results are stored in $P^q(p^q)$, Eq. (73). Third (③), the Q-tree, Fig. 3, is measured N times to sample $\hat{p}(y_i \,|\, \mathcal{C}_\nu)$, Eq. (63), where only leaves present in $\Lambda(p^q)$, Eq. (72), are considered. The results are stored in $P^y(p^q)$, Eq. (74). Finally (④), the stored results can be used for a prediction of the query data via Eq. (70). For a single query data point, this approach requires $\mathcal{O}(1)$ classical memory slots instead of $\mathcal{O}(2^d)$ when also storing the vanishing elements.

in the sense of Eq. (30).

The predicted probability distribution for ith label with $i \in \{1, \ldots, m\}$ consequently reads

$$\hat{p}_{\text{qm}}(y_i \,|\, p^q) \equiv \sum_{\nu=1}^{2^d} \hat{p}^q(\mathcal{C}_\nu \,|\, p^q) \hat{p}(y_i \,|\, \mathcal{C}_\nu), \tag{70}$$

which represents an average of $\hat{p}(y_i \,|\, \mathcal{C}_\nu)$ over all leaves in the sense of an approximation

$$\hat{p}_{\text{qm}}(y_i \,|\, p^q) \approx p(y_i \,|\, p^q) \equiv \sum_{\nu=1}^{2^d} p^q(\mathcal{C}_\nu) p(y_i \,|\, \mathcal{C}_\nu), \tag{71}$$

where we recall Eq. (39). The labels can be obtained in analogy to Eq. (64). The (statistical) uncertainties of Eqs. (63), (69) and (70) are discussed in App. D.

The disadvantage of these two prediction methods is that they presume that the whole estimated label probability distribution $\hat{p}(y_i \,|\, \mathcal{C}_\nu)$, Eq. (61), is available beforehand, which requires $\mathcal{O}(2^d)$ classical memory slots. As an alternative, we propose an *on-demand sampling* prediction method to reduce this requirement to $\mathcal{O}(1)$ classical memory slots. This method consists of four steps, which are sketched in Fig. 7.

First, a probability distribution of query data $p^q(\mathbf{x}^q)$, Eq. (65), is prescribed, which can in particular also represent a single query data point. Second, the corresponding query circuit, Fig. 6, is measured N times to sample the probability distribution of reaching a certain leaf $\hat{p}^q(\mathcal{C}_\nu)$, Eq. (69). All node indices of non-zero elements of this probability distribution are collected in the set

$$\Lambda(p^q) \equiv \left\{ \nu \,|\, \nu \in \{1, \ldots, 2^d\} \wedge \hat{p}^q(\mathcal{C}_\nu \,|\, p^q) > 0 \right\}, \tag{72}$$

where ν iterates over all leaves. For a single query data point, one has $|\Lambda(p^q)| = 1$ such that the set of corresponding non-zero elements

$$P^q(p^q) \equiv \{\hat{p}^q(\mathcal{C}_\nu) \,|\, \hat{p}^q(\mathcal{C}_\nu) \,\forall\, \nu \in \Lambda(p^q)\} \tag{73}$$

requires $\mathcal{O}(1)$ classical memory slots (instead of $\mathcal{O}(2^d)$ when also storing the vanishing elements). Third, the Q-tree of interest, Fig. 3, is measured N times to sample the predicted probability distribution for the ith label $\hat{p}(y_i \,|\, \mathcal{C}_\nu)$, Eq. (63), in such a way that only leaves present in $\Lambda(p^q)$ are considered. That is, for a single query data point, only $\mathcal{O}(1)$ classical memory slots are required to store the resulting set of entries

$$P^y(p^q) \equiv \{\hat{p}(y_i \,|\, \mathcal{C}_\nu) \,|\, \hat{p}(y_i \,|\, \mathcal{C}_\nu) \,\forall\, i \in \{1, \ldots, m\} \,\forall\, \nu \in \Lambda(p^q)\} \tag{74}$$

(instead of $\mathcal{O}(2^d)$ when considering all leaves). Finally, Eq. (70) can be used (omitting the vanishing summands) with the results from Eqs. (73) and (74) to perform the prediction of the prescribed distribution of query data. In total, $\mathcal{O}(1)$ classical memory slots are required for a single query data point. Therefore, an exponential data compression can be achieved for the prediction similar to the exponential data compression for the data encoding from Sec. 4.1.1. The disadvantage of this approach is, of course, that the sampling must be performed anew for each query data based on the evaluation of the query circuit. Thus, the reduction of classical memory is traded off for additional quantum computations.

For a sufficient number of measurements, the on-demand sampling leads to the same predictions as the presampled approach. On noisy quantum hardware, however, the predictions might deviate. For example, one might collect measurements from the query circuit such that $|\Lambda(p^q)| > 1$ despite considering only a single query data point. In this case, choosing the largest probability

$$\tilde{\Lambda}(p^q) \equiv \left\{ \underset{\nu \in \{1,\ldots,2^d\}}{\arg\max} \; \hat{p}^q(\mathcal{C}_\nu \,|\, p^q) \right\} \approx \Lambda(p^q) \tag{75}$$

could serve as a suitable method for noise mitigation. A more detailed discussion of this topic goes beyond the scope of this paper, but can be considered as a possible research direction.

5 Experiments

In the present section, we test the proposed Q-tree approach for quantum representations of decision trees in practice. As our study example, we consider two data sets. First, the *tic-tac-toe endgame* data set [59] from the UCI machine learning repository [60], which we use for experiments on a quantum computing simulator (on classical hardware), and second, a simple toy data set, which we use to run experiments on actual quantum hardware. In the following, we start by describing the data. Subsequently, we present the experimental results on the simulator and the hardware, respectively.

5.1 Data

We consider two data sets for our experiments. The first data set is the tic-tac-toe endgame data set [59], which contains the complete set of possible board configurations at the end of tic-tac-toe games. Each of the 958 instances corresponds to one legal tic-tac-toe endgame board. A board is encoded by a nine-dimensional vector of ternary attributes $\mathbf{x}' \in \{0, 1, 2\}^9$, which determine whether each of the nine board fields is taken by one of the two players or left blank (0 represents starting player, 1 represents the other player and 2 represents a blank field) with the allocation shown in Fig. 8. We convert $\mathbf{x}'$ into a 15-dimensional feature vector $\mathbf{x} \in \mathbb{B}^{15}$ (i. e., $k = 15$) according to

$$\sum_{i=0}^{9} x'_{9-i} 3^i = \sum_{i=0}^{15} x_{15-i} 2^i. \tag{76}$$

In addition, a binary label $\mathbf{y} \in \mathbb{B}^1$ (i. e., $m = 1$) is assigned to each board instance to decide the player who won the game ($y_1 = 1$ represents a victory of the starting player and $y_1 = 0$ a victory of the other player), where a victory is achieved by a "three-in-a-row."

x'_1	x'_2	x'_3
x'_4	x'_5	x'_6
x'_7	x'_8	x'_9

Figure 8: Allocation of the nine ternary attributes $x'_1, \ldots, x'_9 \in \{0, 1, 2\}^9$ to the nine tic-tac-toe board game fields. The attributes determine whether a field $i \in \{1, \ldots, 9\}$ is taken by one of the two players ($x'_i = 0$ for the starting player or $x'_i = 1$ for the other player) or left blank ($x'_i = 2$).

The second data set we consider is a toy data set, which consists of five seven-dimensional binary feature vectors (i. e., $k = 7$) and five corresponding binary labels (i. e., $m = 1$). In total, the data set reads

$$\mathbf{X} \equiv \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } \mathbf{Y} \equiv \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \tag{77}$$

where each row represents a data point and each column a feature vector and label, respectively. By definition, $y_1 = 1$ if $\mathbf{x}$ contains an odd number of ones and $y_1 = 0$ otherwise (i. e., the label represents an *exclusive or* operation over all features). In particular, the label depends only on the first three features, whereas the last four features carry no information.

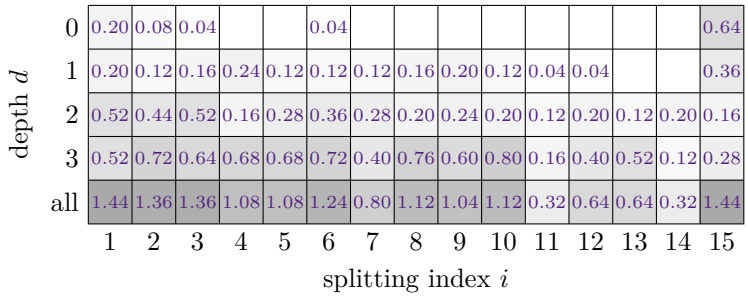
5.2 Simulator

We use the quantum computing simulator *Qiskit* (Quantum Information Software Kit) [61] to evaluate quantum circuits on classical hardware. In particular, we empirically test Q-trees using the data-based decision problem to predict the winner of a tic-tac-toe match $\mathbf{y} \in \mathbb{B}^1$ based on the board configuration $\mathbf{x} \in \mathbb{B}^{15}$ as described in Sec. 5.1. For this purpose, the tic-tac-toe endgame data set is randomly split into a training data set $\mathbf{D}_{\text{train}}$, Eq. (1), and a test data set $\mathbf{D}_{\text{test}}$ of equal size (i. e., 479 instances each) such that both splitted data sets contain the same proportion of labels. We use the training data set to induce decision trees and the test data set to verify the respective tree performances as explained in the following in more detail.

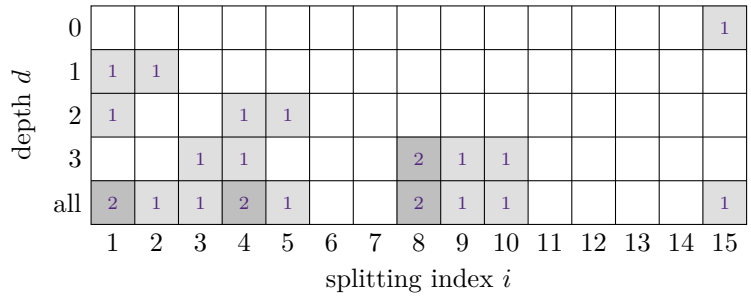
To begin with, we induce a decision tree of maximum depth $d = 3$ as described in Sec. 4.2 using the genetic algorithm from App. B with the hyperparameters $\theta = (16, 20, 3, 0.3, 0.5, 0.15, 1, 1)$, Eq. (101), and $N = 10^6$ measurements. We repeat the induction 25 times with different random seeds. As a result, we obtain 25 Q-trees $QT^1_{\text{sim}}, \ldots, QT^{25}_{\text{sim}}$ (each parameterized by a corresponding compressed decision configuration, Eq. (12)). The mean distribution of splitting feature indices in each depth is shown in Fig. 9a. We find that the splitting index 15 occurs most often for the root, whereas the most often occurring splitting indices over all depths are 1 and 15.

A final sampling with $N = 10^6$ measurements is performed for each Q-tree to obtain an estimate for the label probability distribution $\hat{p}(y_i \,|\, \mathcal{C}_\nu)$, Eq. (61), for all leaves ν, Eq. (8), with the corresponding set of conditions $\mathcal{C}_\nu$, Eq. (28). Based on this estimate, predictions of query data can be performed using Eq. (63). In case that we find zero samples for a particular leaf (i. e., $n(\mathcal{C}_\nu) = 0$), the probability distribution $p(y_1 \,|\, \mathcal{C}_\nu)$, Eq. (61), is undefined. To be able to still make predictions, we assign the agnostic default value $p(y_1 \,|\, \mathcal{C}_\nu) = \frac{1}{2}$. Furthermore, to resolve the resulting ambiguity of $\hat{y}_1$, Eq. (64), we instead predict the majority label present in the training data set.

For comparison purposes, we employ *scikit-learn* [62] to induce a classical decision tree T_{cl} of the same maximum depth using a top-down approach with information entropy as a splitting criterion. The resulting tree is shown in Fig. 10b. The corresponding distribution of splitting feature indices in each depth is visualized in Fig. 9b. Clearly, there is a close resemblance between the distributions in Fig. 9a and Fig. 9b, but there are also certain differences. For example, the splitting index for the root is 15 in Fig. 9b, which corresponds to the majority of splitting indices in Fig. 9a. On the other hand, the most often occurring splitting indices over all depths in Fig. 9b are 1, 4, and 8 as opposed to 1 and 15 in Fig. 9a. Here and in the following, we use the definition imposed by



(a) Mean splitting feature index distribution for the Q-trees $QT^1_{\text{sim}}, \ldots, QT^{25}_{\text{sim}}$.



(b) Splitting feature index distribution for the classical decision tree T_{cl}, which is also shown in Fig. 10b.

Figure 9: Distribution of splitting feature indices for considered decision trees. Each column represents a feature index $i \in \{1, \ldots, 15\}$ and each row a depth $d \in \{0, 1, 2, 3\}$. The values in each cell denote the number of splitting decisions with feature index i in depth d, where we omit zero values. The last row represents the number of splitting decisions cumulated over all depths. Darker backgrounds of the cells indicate higher numbers.

scikit-learn that balanced label probabilities $p(y_1 = 0 \,|\, \mathcal{C}_\nu) = p(y_1 = 0 \,|\, \mathcal{C}_\nu) = \frac{1}{2}$, Eq. (31), lead to a prediction $y_1 = 0$ for T_{cl}.

As a primary performance metric for the decision trees, we consider the balanced accuracy

$$\text{bac} \equiv \frac{1}{2}\left(\frac{\text{tp}_0}{\text{tp}_0 + \text{fn}_0} + \frac{\text{tn}_0}{\text{tn}_0 + \text{fp}_0}\right) = \frac{1}{2}\left(\frac{\text{tp}_1}{\text{tp}_1 + \text{fn}_1} + \frac{\text{tn}_1}{\text{tn}_1 + \text{fp}_1}\right), \tag{78}$$

which is based on the number of true positives

$$\text{tp}_b \equiv |\{\mathbf{d} \,|\, (\mathbf{x}^q, (y_1)) = \mathbf{d} \in \mathbf{D} \wedge y_1 = b \wedge \hat{y}_1(\mathbf{x}^q) = b\}|, \tag{79}$$

the number of true negatives

$$\text{tn}_b \equiv |\{\mathbf{d} \,|\, (\mathbf{x}^q, (y_1)) = \mathbf{d} \in \mathbf{D} \wedge y_1 = 1 - b \wedge \hat{y}_1(\mathbf{x}^q) = 1 - b\}|, \tag{80}$$

the number of false positives

$$\text{fp}_b \equiv |\{\mathbf{d} \,|\, (\mathbf{x}^q, (y_1)) = \mathbf{d} \in \mathbf{D} \wedge y_1 = 1 - b \wedge \hat{y}_1(\mathbf{x}^q) = b\}|, \tag{81}$$

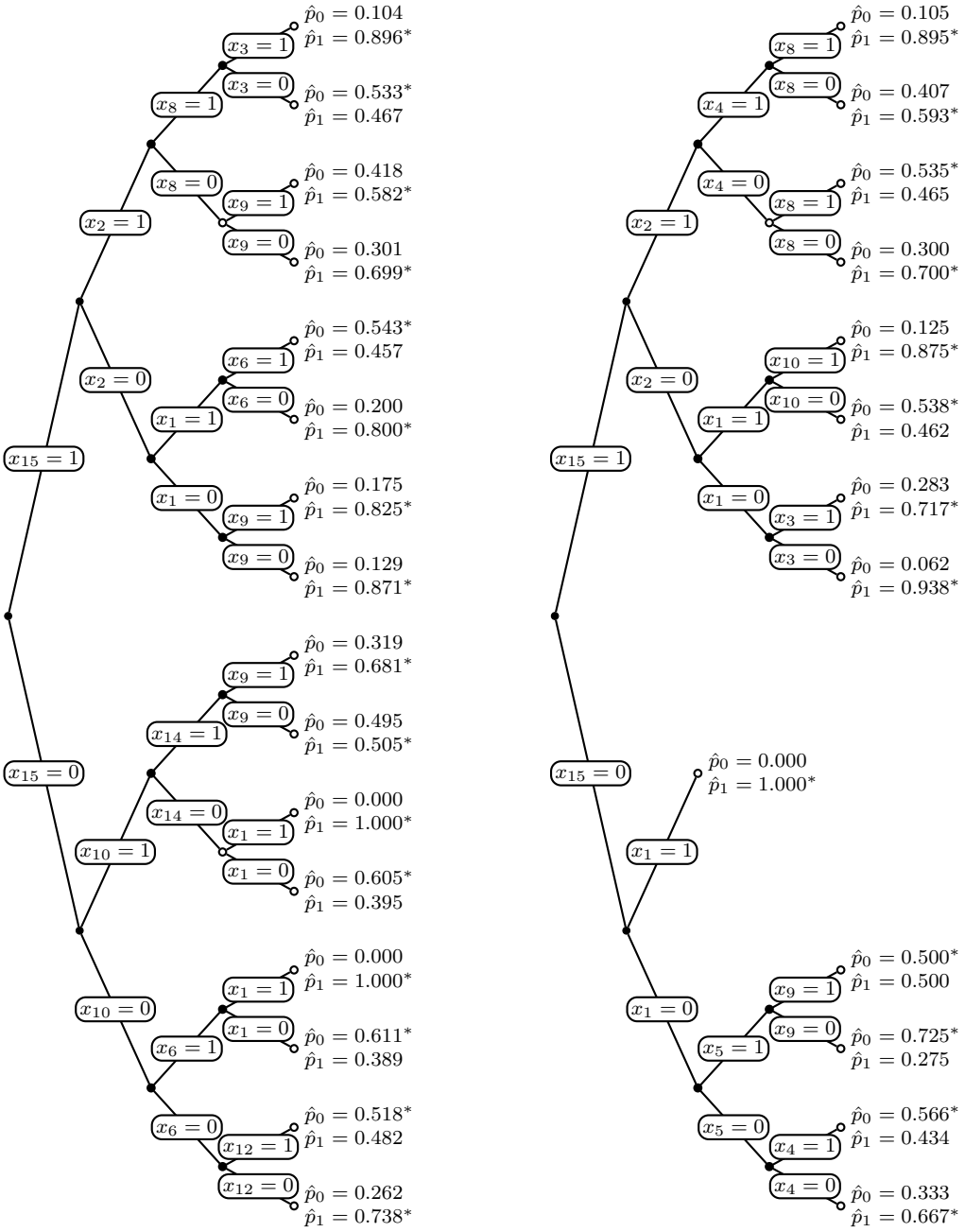
and the number of false negatives

$$\text{fn}_b \equiv |\{\mathbf{d} \,|\, (\mathbf{x}^q, (y_1)) = \mathbf{d} \in \mathbf{D} \wedge y_1 = b \wedge \hat{y}_1(\mathbf{x}^q) = 1 - b\}|, \tag{82}$$

respectively, where $b \in \mathbb{B}$ denotes the labels treated as positives and $\mathbf{D}$ the data set of interest. Here, $\hat{y}_1(\mathbf{x}^q) \in \mathbb{B}$ represents a prediction of the label query data $\mathbf{x}^q \in \mathbb{B}^{15}$. For the Q-tree we use Eq. (64) to perform predictions.

To obtain the "best" decision tree out of our induced sequence of Q-trees, we choose the tree with the highest balanced accuracy with respect to the training data set $\mathbf{D}_{\text{train}}$. The resulting tree QT_{sim} is associated with the decision configuration

$$C_{\text{sim}} = ((15), (10, 2), (6, 14, 15, 8), (12, 15, 15, 9, 9, 6, 9, 4)) \tag{83}$$



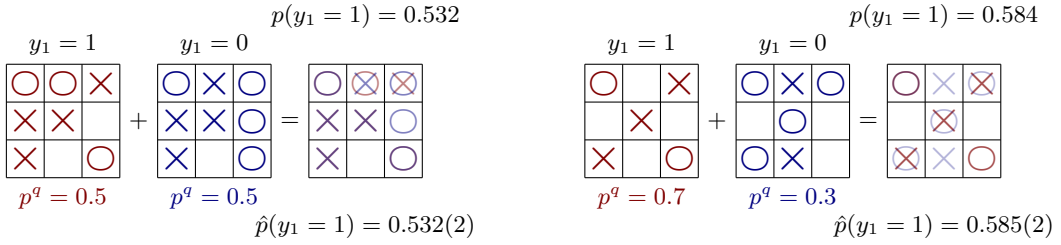
(a) Best Q-tree QT_{sim} with the compressed decision configuration C_{sim}, Eq. (83), induced via a genetic algorithm. Here, $\hat{p}_b \equiv \hat{p}(y_1 = b\,|\,\mathcal{C}_\nu)$ denote the estimated label probabilities for $N = 10^6$ measurements, Eq. (61), which are also shown in Fig. 13a.

(b) Classical decision tree T_{cl} induced via scikit-learn using a top-down approach with a splitting criterion based on the information entropy. Here, $\hat{p}_b \equiv p(y_1 = b\,|\,\mathcal{C}_\nu)$ denote the exact label probabilities, Eq. (31).

Figure 10: Induced decision trees with maximum depth $d = 3$ for the tic-tac-toe training data set $\mathbf{D}_{\text{train}}$, where all quantum computations have been performed on a simulator. Starting from the root on the left, we show the decisions $x_i = b$ along all paths between the internal nodes (•) to the leaves on the right (⚬), where $i \in \{1, \ldots, 15\}$ and $b \in \mathbb{B}$. For each leaf, we also show the probability $\hat{p}_b$ of predicting $y_1 = b$. The predicted label is indicated with a star (*) at the corresponding probability.

(a) correct **(b)** incorrect **(c)** correct **(d)** incorrect

Figure 11: Four exemplary predictions using the Q-tree QT_{sim}, Eq. (83) and Fig. 10a. For each of the four examples, we visualize the query data $\mathbf{x}^q = \mathbf{x}$ in form of the corresponding board configuration based on Eq. (76), where ✕ and ○ represent a field taken by the starting player and the other player, respectively. Also shown is the true label $y_1 = b$ with $b \in \mathbb{B}$, where $b = 1$ represents a victory of the starting player and $b = 0$ a victory of the other player. The predicted probability of the true label is denoted by $\hat{p}(y_1 = b) \equiv \hat{p}_{\text{cl}}(y_1 = b \,|\, \mathbf{x}^q)$, Eq. (63). The corresponding bit string $\bar{x}_0\bar{x}_1\bar{x}_2\bar{x}_3$ represents the traversed path in the tree, Eq. (38). We indicate for each example whether the prediction is correct or incorrect using Eq. (40).



(a) balanced **(b)** imbalanced

Figure 12: Two exemplary uncertain query data predictions using the Q-tree QT_{sim}, Eq. (83) and Fig. 10a. Using the same notation as in Fig. 11, we show an uncertain overlay of two boards weighted by balanced and imbalanced probabilities p^q, respectively. The opacity of the resulting overlay on the right (with respect to the two colors ✕/○ and ✕/○) reflects the corresponding values of p^q (the more transparent, the smaller). For each example, we show the true labels of the query components $y_1 = b$ with $b \in \mathbb{B}$, the true predicted probability distribution for the query label $p(y_1 = b) \equiv p(y_1 = b \,|\, p^q)$, Eq. (71), as well as its prediction $\hat{p}(y_1 = b) \equiv \hat{p}_{\text{qm}}(y_1 = b \,|\, p^q)$, Eq. (70), with the standard deviation $\sigma\hat{p}_{\text{qm}}(y_1 \,|\, p^q)$, Eq. (132), in brackets. All quantum computations have been performed on a simulator.

and shown Fig. 10a. Four exemplary predictions for QT_{sim} based on Eq. (63) are presented in Fig. 11. Two correct and two incorrect predictions are used for this purpose. Since the predictions are performed on the basis of previously sampled probability distributions, no additional quantum computations are necessary. In addition, we also demonstrate two uncertain query predictions using $\hat{p}_{\text{qm}}(y_i \,|\, p^q)$, Eq. (70), in Fig. 12. They require additional quantum computations for the estimation of $\hat{p}^q(\mathcal{C}_\nu \,|\, p^q)$, Eq. (69). The predicted probability distribution from the sampling, Eq. (70), coincides with the true probability distribution calculated from the data proportions, Eq. (71). However, since the two uncertain queries represent a mixture of boards with opposite labels, we cannot assess whether or not the resulting predictions are correct in the sense of a correct classification.

Next, we perform a direct comparison of the classification performance between $QT^1_{\text{sim}}, \ldots, QT^{25}_{\text{sim}}$ and T_{cl}. In addition to the balanced accuracy, Eq. (78), we also consider the unbalanced accuracy

$$\text{acc} \equiv \frac{\text{tp}_0 + \text{tn}_0}{\text{tp}_0 + \text{tn}_0 + \text{fp}_0 + \text{fn}_0} = \frac{\text{tp}_1 + \text{tn}_1}{\text{tp}_1 + \text{tn}_1 + \text{fp}_1 + \text{fn}_1}, \tag{84}$$

the precision (fraction of relevant instances among the retrieved instances)

$$\text{pre}_b \equiv \frac{\text{tp}_b}{\text{tp}_b + \text{fp}_b}, \tag{85}$$

Table 1: Classification performance metrics, Eqs. (78) and (84) to (87), of the Q-trees $QT_{\text{sim}}^1, \ldots, QT_{\text{sim}}^{25}$ and the classical decision tree T_{cl}, respectively. We list the mean and standard deviation (in brackets) over all Q-trees, the best metric over all trees and the metrics of the exemplary Q-tree QT_{sim}, Eq. (83) and Fig. 10a. The best results (with respect to two digits) are highlighted in bold.

	bac	acc	pre_0	pre_1	rec_0	rec_1	f1_0	f1_1
Training data								
Mean Q-tree metric	0.65(2)	0.70(1)	0.58(3)	0.75(2)	0.50(7)	0.80(5)	0.53(4)	0.77(1)
Best Q-tree metric	0.68	**0.72**	**0.66**	0.78	0.61	**0.90**	0.58	**0.81**
Q-tree QT_{sim}	0.68	0.71	0.57	0.78	0.59	0.77	0.58	0.77
Classical tree T_{cl}	**0.70**	0.71	0.58	**0.80**	**0.64**	0.75	**0.61**	0.78
Test data								
Mean Q-tree metric	0.63(3)	0.69(2)	0.57(4)	0.74(2)	0.44(7)	0.82(4)	0.50(5)	0.78(2)
Best Q-tree metric	**0.68**	**0.72**	**0.63**	**0.78**	**0.58**	**0.88**	**0.58**	**0.80**
Q-tree QT_{sim}	0.62	0.68	0.55	0.73	0.43	0.81	0.48	0.77
Classical tree T_{cl}	0.67	**0.72**	0.61	0.76	0.52	0.83	0.56	0.79

the recall (fraction of relevant instances that were retrieved)

$$\text{rec}_b \equiv \frac{\text{tp}_b}{\text{tp}_b + \text{fn}_b}, \tag{86}$$

and the balanced F-score (harmonic mean of precision and recall)

$$\text{f1}_b \equiv \frac{2\text{pre}_b\text{rec}_b}{\text{pre}_b + \text{rec}_b}, \tag{87}$$

respectively, with $b \in \mathbb{B}$, where we recall Eqs. (79) to (82). The performance metrics are determined for the training and test data set (i. e., $\mathbf{D} \in \{\mathbf{D}_{\text{train}}, \mathbf{D}_{\text{test}}\}$).

The results are listed in Tab. 1. We find that for the training data, certain best metrics of the Q-trees (acc, pre_0, rec_1, and f1_1) are superior to the corresponding metrics of the classical tree T_{cl}, whereas others (bac, pre_1, rec_0, and f1_0) are inferior. For the test data, the best metrics of the Q-trees are always superior or at least as good as the corresponding metrics of the classical tree. In all cases, the mean Q-tree metrics and the metrics of the exemplary Q-tree QT_{sim}, Eq. (83), are worse than or equal to the corresponding metrics of the classical tree.

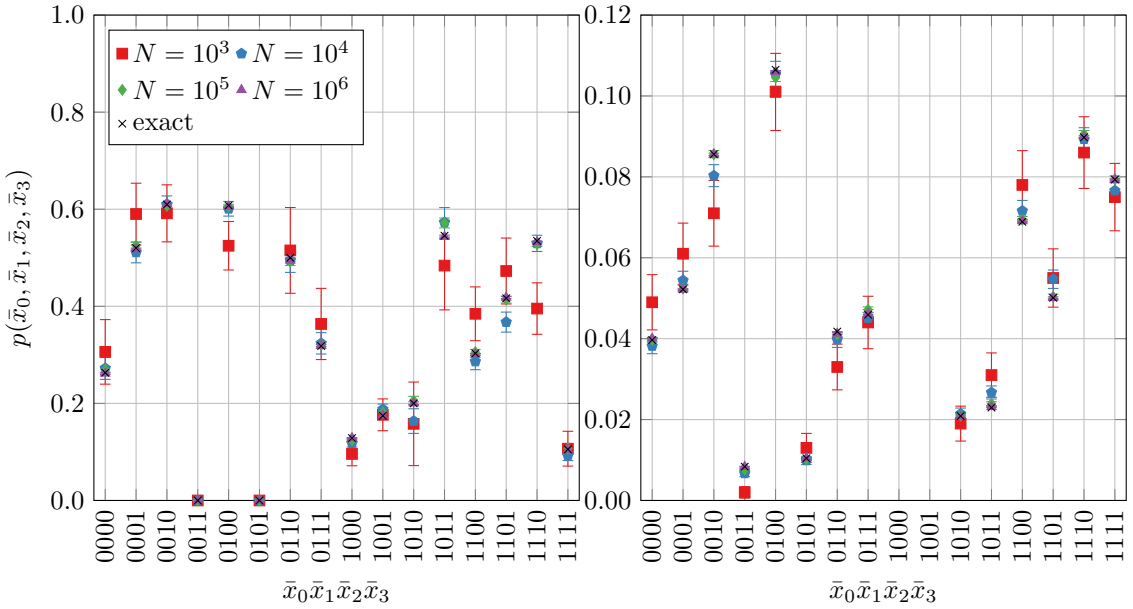
To quantify the performance of the tree induction, we consider a set of randomly generated decision trees. For this purpose, we uniformly draw 25 decision configurations, Eq. (12). Subsequently, we perform a sampling with $N = 10^6$ measurements for each corresponding Q-tree to obtain an estimate for the label probability distribution. We determine the balanced training and test accuracy, Eq. (78), for each tree and obtain a mean and standard deviation (in brackets) of 0.53(4) for both test and training data. This result is significantly worse than the mean balanced accuracy over all induced trees listed in Tab. 1. We consequently conclude that our induction method is clearly superior to a random selection of trees for this particular example.

Finally, we compare the effect of using different numbers of experiments $N \in \{10^3, 10^4, 10^5, 10^6\}$ to estimate the probability distribution of the decision set $\hat{p}(\mathcal{C}_\nu)$, Eq. (60), and the corresponding label probability distribution $\hat{p}(y_i \,|\, \mathcal{C}_\nu)$, Eq. (61), for the Q-tree QT_{sim}. We compare the estimates with the exact probability distributions based on data proportions $p(\mathcal{C}_\nu)$, Eq. (30), and $p(\mathbf{y} \,|\, \mathcal{C}_\nu)$, Eq. (31), respectively, and also consider the statistical uncertainties as discussed in App. D.

The results are shown in Fig. 13. As expected, the sampled probabilities converge to the exact results for a sufficiently high number of experiments, whereas their standard deviations decrease with an increasing number of experiments. Moreover, the standard deviations of vanishing probabilities also vanish.

5.3 Quantum hardware

A major challenge of quantum algorithms is to run them on noisy intermediate-scale quantum (NISQ) devices [63]. To perform quantum hardware experiments, we remotely access a physical



(a) Label probabilities from Q-tree samples $p(\bar{x}_0, \bar{x}_1, \bar{x}_2, \bar{x}_3) \equiv \hat{p}(y_1 = 0 \,|\, \mathcal{C}_{\nu_4(\bar{x}_0,\bar{x}_1,\bar{x}_2,\bar{x}_3)})$, Eq. (61), with standard deviation (one sigma) $\hat{\sigma}\hat{p}'(k, m, N)$, Eq. (128), and from exact calculations $p(\bar{x}_0, \bar{x}_1, \bar{x}_2, \bar{x}_3) \equiv p(y_1 = 0 \,|\, \mathcal{C}_{\nu_4(\bar{x}_0,\bar{x}_1,\bar{x}_2,\bar{x}_3)})$, Eq. (31), respectively.

(b) Probabilities of the decision set from Q-tree samples $p(\bar{x}_0, \bar{x}_1, \bar{x}_2, \bar{x}_3) \equiv \hat{p}(\mathcal{C}_{\nu_4(\bar{x}_0,\bar{x}_1,\bar{x}_2,\bar{x}_3)})$, Eq. (60), with standard deviation (one sigma) $\hat{\sigma}\hat{p}(n, N)$, Eq. (120), and from exact calculations $p(\bar{x}_0, \bar{x}_1, \bar{x}_2, \bar{x}_3) \equiv p(\mathcal{C}_{\nu_4(\bar{x}_0,\bar{x}_1,\bar{x}_2,\bar{x}_3)})$, Eq. (30), respectively. We use the same symbols as in (a).

Figure 13: Sampled probability distributions of the Q-tree QT_{sim}, Eq. (83) and Fig. 10a, using $N \in \{10^3, 10^4, 10^5, 10^6\}$ measurements. All quantum computations have been performed on a simulator. For comparison, we also show the exact probability distributions based on the data proportions.

quantum computer via Qiskit using the cloud-based quantum computing service provided by *IBM Quantum* [64]. This service allows online requests for quantum experiments using a high-level quantum circuit model of computation [65]. The quantum experiments are then realized and executed sequentially on physical hardware that operates on superconducting transmon qubits. Specifically, we access the quantum computer *ibmq_ehningen* (version 2.2.1), which represents a so-called *Falcon r4* processor with 27 qubits and a quantum volume [66] of 32.

A quantum experiment is in this context defined as a circuit to be evaluated on the quantum computer, whereas the returned result consist of a histogram of counts for all measured qubits. The quantum computer we use is only capable of executing a specific set of gates (NOT, R_z, $\sqrt{\mathrm{NOT}}$, and CNOT) and has a limited connectivity of qubits (i. e., CNOT gates can only be applied to specific pairs of qubits) as sketched in Fig. 14. In a process called *transpilation*, Qiskit can algorithmically translate a general circuit into one that can be run on a specific hardware. We use this functionality to realize our quantum experiments for the Q-tree. Furthermore, we make use of Qiskit's built-in state preparation approach [34] to find the data encoding operator $\hat{U}_{\text{data}}(\mathbf{D}_{\text{train}})$, Eq. (42).

In total, a Q-tree of maximum depth d consists of $k + m$ qubits and exhibits a complexity of $\mathcal{O}(2^{k+m} + d2^d)$ gates. It is possible to reduce this complexity by only encoding data in such qubits that are actually measured. This effectively corresponds to performing the marginalization of $p(\mathbf{x}, \mathbf{y})$, Eq. (21), beforehand and leads to a circuit consisting of $d{+}1{+}m$ qubits and $\mathcal{O}(2^d(d{+}2^m))$ gates. In addition, the first SWAP gate can be omitted by a suitable re-ordering of the data (i. e., by assigning the corresponding feature of the root decision to the first qubit). The data preparation consequently depends on the tree structure in these cases. To simplify our circuits, we make use of these complexity reductions in the following.

The tic-tac-toe endgame data set considered in Sec. 5.2 for the simulator turns out to be too complex for the actual quantum hardware. Therefore, we instead employ the simpler toy data set, Eq. (77), where we use the total data set for both training and testing purposes. In analogy

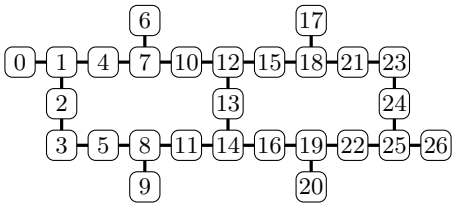
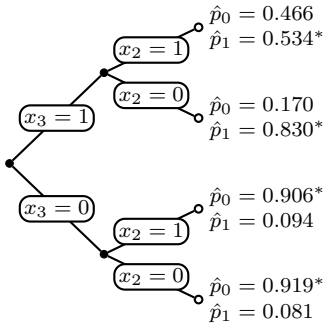
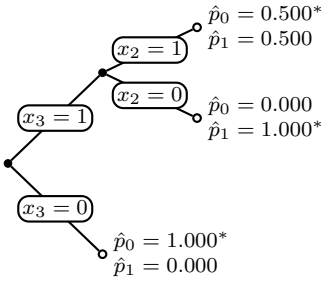


Figure 14: Connectivity diagram of the quantum computer *ibmq_ehningen* with 27 qubits. Qubits are shown as boxes with their respective hardware index $\{0, \ldots, 26\}$ as provided by IBM Quantum. Pairs of qubits that support a mutual CNOT gate are connected with lines.



(a) Best Q-tree QT_{qc} with the compressed decision configuration C_{qc}, Eq. (88), induced via a genetic algorithm.



(b) Classical decision tree T'_{cl} induced via scikit-learn using a top-down approach with a splitting criterion based on the information entropy.

Figure 15: Induced decision trees with maximum depth $d = 1$ for the toy data set, Eq. (77), where all quantum computations have been performed on actual quantum hardware. We use the same notation as in Fig. 10.

to our simulator experiments, we start by inducing a decision tree of maximum depth $d = 1$ as described in Sec. 4.2 using the genetic algorithm from App. B. We use the hyperparameters $\theta = (31, 6, 4, 0.5, 0.5, 0.5, 1, 1)$, Eq. (101), and $N = 8192$ measurements (which is the maximum number of measurements – so-called *shots* – that can be performed on the chosen hardware in a single run). We repeat the induction 10 times with different random seeds and consequently obtain 10 Q-trees $QT^1_{\mathrm{qc}}, \ldots, QT^{10}_{\mathrm{qc}}$. Again, we perform a final sampling with $N = 8192$ measurements to obtain an estimate for the label probability distribution $\hat{p}(y_i \,|\, \mathcal{C}_\nu)$, Eq. (61). In addition, we use scikit-learn to induce a classical decision tree T'_{cl} of the same maximum depth for comparison purposes, which is shown in Fig. 15b.

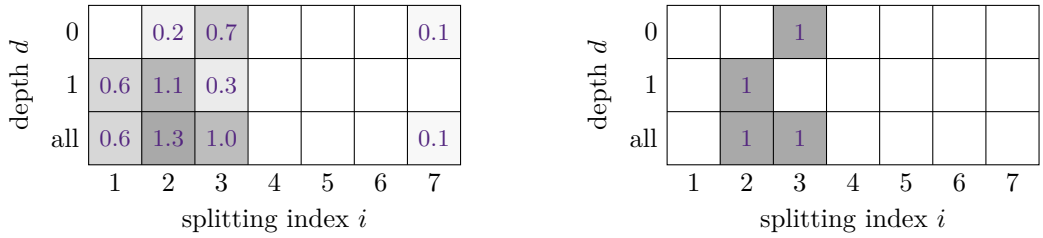
We visualize the mean distribution of splitting feature indices in each depth in Fig. 16 and find a close resemblance between the distributions in Fig. 16a and Fig. 16b. Furthermore, the first three splitting indices, which refer to informative features, are chosen over the remaining splitting indices, which refer to uninformative features (in all except for one case in Fig. 16a). The most often occurring splitting index over all depths is 2 in Fig. 16a, whereas in Fig. 16a the splitting indices 2 and 3 are both occurring once in the tree.

We choose the Q-tree with the highest balanced accuracy, Eq. (78), for demonstration purposes. This "best" Q-tree QT_{qc} is associated with the decision configuration

$$C_{\mathrm{qc}} = ((3), (2, 2)) \tag{88}$$

and shown in Fig. 15a. The structure of this Q-tree is particularly equivalent to the structure of the corresponding classical decision tree T'_{cl} except for the additional split at depth $d = 1$. We compare the previously introduced classification performance metrics of the Q-trees $QT^1_{\mathrm{qc}}, \ldots, QT^{10}_{\mathrm{qc}}$ and the classical decision tree T_{cl}.

The results are shown in Tab. 2. We find that all best metrics of the Q-trees are superior to the corresponding metrics of the classical tree T'_{cl}. Furthermore, the majority of the metrics



(a) Mean splitting feature index distribution for the Q-trees $QT_{\mathrm{qc}}^{1}, \ldots, QT_{\mathrm{qc}}^{10}$.

(b) Splitting feature index distribution for the classical decision tree T'_{cl}, which is also shown in Fig. 15b.

Figure 16: Distribution of splitting feature indices for considered decision trees in analogy to Fig. 9.

Table 2: Classification performance metrics, Eqs. (78) and (84) to (87), of the Q-trees $QT_{\mathrm{qc}}^{1}, \ldots, QT_{\mathrm{qc}}^{10}$, the exemplary Q-tree QT_{qc}, Eq. (88), and the classical decision tree T'_{cl}, respectively, in analogy to Tab. 1. When calculating the mean and standard deviation, we ignore all metrics that are not well-defined due to a division by zero. The best results (with respect to two digits) are highlighted in bold.

	bac	acc	pre_0	pre_1	rec_0	rec_1	$\mathrm{f1}_0$	$\mathrm{f1}_1$
Mean Q-tree metric	0.71(11)	0.76(8)	0.74(10)	0.96(11)	0.97(10)	0.45(27)	0.83(4)	0.68(4)
Best Q-tree metric	**0.83**	**0.80**	**1.00**	**1.00**	**1.00**	**1.00**	**0.86**	**0.80**
Q-tree QT_{qc}	**0.83**	**0.80**	**1.00**	0.67	0.67	**1.00**	0.80	**0.80**
Classical tree T'_{cl}	0.75	**0.80**	0.75	**1.00**	**1.00**	0.50	**0.86**	0.78

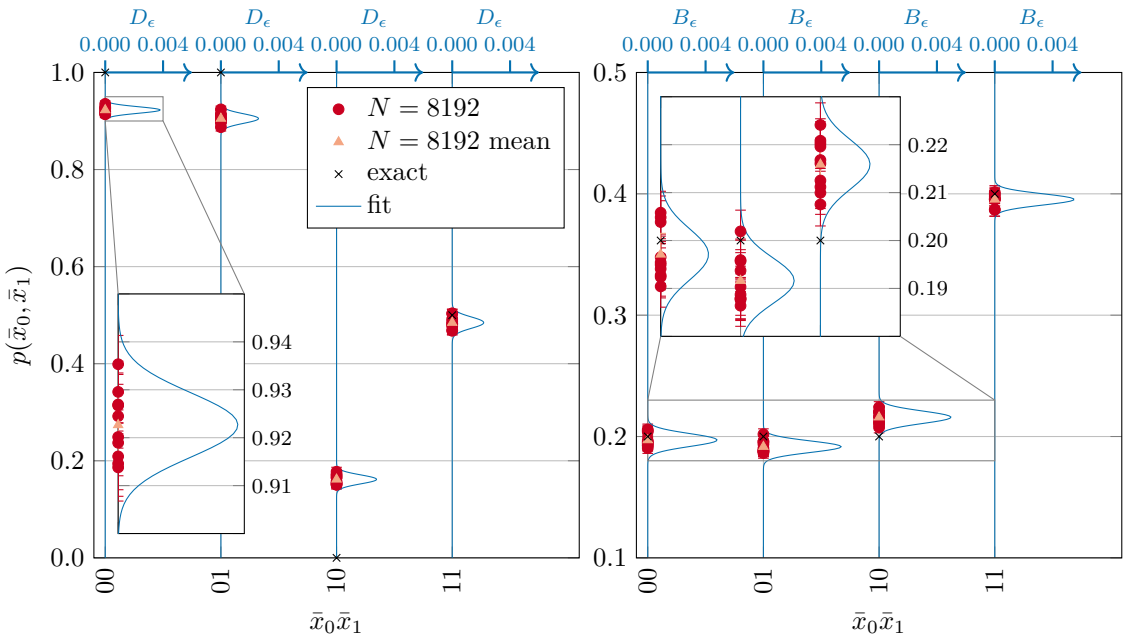
of the exemplary Q-tree QT_{qc} (bac, acc, pre_0, rec_1, and $\mathrm{f1}_1$) are better than or equal to the corresponding metrics of the classical tree, whereas the others (pre_0, rec_1, and $\mathrm{f1}_1$) are worse. Due to their similar structure, the only difference between the predictions of QT_{qc} and T'_{cl} comes from the path (or bit string) $\bar{x}_0\bar{x}_1 = 11$. For the corresponding leaf, the Q-tree predicts $y_1 = 1$, whereas the classical tree estimates balanced label probabilities that lead to a prediction $y_1 = 0$ (according to the scikit-learn default behavior). All advantages of QT_{qc} are therefore a consequence of the noise from the quantum sampling process. Given these facts, the apparent superiority that can be inferred from Tab. 2 is questionable, and there is also no fundamental reason why our quantum representation should lead to better predictions than the classical representation. Given the simplicity of the problem, this is no surprise. However, our results at least indicate that the quantum representation running on a NISQ device is (for this particular example) not significantly inferior to the classical representation.

To verify that our tree induction approach is superior to a random guess, we create a set of 10 randomly generated decision trees by uniformly drawing decision configurations, Eq. (12), and subsequently sample with $N = 8192$ measurements to obtain an estimate for the label probability distribution. The resulting a mean and standard deviation (in brackets) for the balanced accuracy, Eq. (78), over all trees reads 0.51(2) and is therefore significantly worse than the mean balanced accuracy over all induced trees listed in Tab. 2.

Next, we consider the estimations for the probability distributions for the best Q-tree QT_{qc} in analogy to our experiments on the simulator. The results are shown in Fig. 17. We find that some of the sampled probabilities deviate significantly from the corresponding exact probabilities. We suppose that these deviations result from the hardware-induced noise.

To further quantify our observation, we take a closer look at the deviations. For this purpose, we assume that statistical errors of the estimated probability distributions are negligible due to a sufficiently large sampling size in comparison with the hardware-induced noise as discussed in App. D. Consequently, we can make use of the approximations in Eqs. (143) and (144), respectively. We furthermore presume that the hardware-induced noise, Eqs. (139) and (140), can be described by a truncated normal distribution $\mathcal{N}_0^1(x; \mu, \sigma)$ within the interval $(0, 1)$ with mean μ and standard deviation σ, where $x \in (0, 1)$. Consequently, we arrive at the effective probability distributions

$$B_\epsilon(n; N, p) \approx \mathcal{N}_0^1\left(\frac{n}{N}; p + \mu_\epsilon, \sigma_\epsilon\right) \tag{89}$$



(a) Label probabilities from Q-tree samples $p(\bar{x}_0, \bar{x}_1) \equiv \hat{p}(y_1 = 0 \,|\, \mathcal{C}_{\nu_2(\bar{x}_0,\bar{x}_1)})$, Eq. (61), with standard deviation (one sigma) $\hat{\sigma}\hat{p}'(k, m, N)$, Eq. (128), and from exact calculations $p(\bar{x}_0, \bar{x}_1) \equiv p(y_1 = 0 \,|\, \mathcal{C}_{\nu_2(\bar{x}_0,\bar{x}_1)})$, Eq. (31), respectively. The effective probability distributions ("fit") corresponds to $D_\epsilon(k, m; N, p', q)$, Eq. (90), with the parameters μ'_ϵ and σ'_ϵ shown in Tab. 3 and is plotted with respect to one of the four separate axes shown on top. An inset plot shows the results for $\bar{x}_0\bar{x}_1 = 00$ in more detail.

(b) Probabilities of the decision set from Q-tree samples $p(\bar{x}_0, \bar{x}_1) \equiv \hat{p}(\mathcal{C}_{\nu_2(\bar{x}_0,\bar{x}_1)})$, Eq. (60), with standard deviation (one sigma) $\hat{\sigma}\hat{p}(n, N)$, Eq. (120), and from exact calculations $p(\bar{x}_0, \bar{x}_1) \equiv p(\mathcal{C}_{\nu_2(\bar{x}_0,\bar{x}_1)})$, Eq. (30), respectively. The effective probability distributions ("fit") corresponds to $B_\epsilon(n; N, p)$, Eq. (89), with the parameters μ_ϵ and σ_ϵ shown in Tab. 3 and is plotted with respect to one of the four separate axes shown on top. An inset plot shows the results for $\bar{x}_0\bar{x}_1 = 00$, $\bar{x}_0\bar{x}_1 = 01$, and $\bar{x}_0\bar{x}_1 = 10$ in more detail. We use the same symbols as in (a).

Figure 17: Sampled probability distributions of the Q-tree QT_{qc}, which is also shown in Fig. 15a, using $N = 8192$ measurements. We use a similar notation as in Fig. 13. All quantum computations have been performed on actual quantum hardware. For comparison, we also show the exact probability distributions based on the data proportions as well as estimations for the hardware-induced noise ("fit") based on the effective probability distributions from Eqs. (89) and (90) with the fitted parameters from Tab. 3. For the effective probability distributions, a separate axis is provided on top of the plot for each bit string $\bar{x}_0\bar{x}_1$.

Table 3: Fitted parameters for the effective probability distributions from Eqs. (89) and (90), respectively. The expectation value shifts μ_ϵ and μ'_ϵ represent hardware-induced systematic errors, whereas the standard deviations σ_ϵ and σ'_ϵ represent hardware-induced statistical uncertainties. The resulting effective probability distributions are also shown in Fig. 17.

$\bar{x}_0\bar{x}_1$	μ_ϵ	σ_ϵ	μ'_ϵ	σ'_ϵ
00	-0.0029	0.0053	-0.0773	0.0067
01	-0.0084	0.0048	-0.0950	0.0098
10	$+0.0159$	0.0051	$+0.1617$	0.0092
11	-0.0046	0.0047	-0.0154	0.0117

and

$$D_\epsilon(k, m; N, p', q) \approx \mathcal{N}_0^1\left(\frac{k}{m}; p' + \mu'_\epsilon, \sigma'_\epsilon\right), \tag{90}$$

respectively. Here we introduce the expectation value shifts μ_ϵ and μ'_ϵ, which represent hardware-induced systematic errors, and the standard deviations σ_ϵ and σ'_ϵ, which represent hardware-induced statistical uncertainties. These parameters can be obtained by fitting Eqs. (89) and (90) to the estimated probability distributions from the measurements.

The fitted parameters are listed in Tab. 3. The resulting effective probability distributions are also shown in Fig. 17. As expected, we find both non-vanishing statistical and systematic errors for all bit strings. The systematic errors are neither strictly positive or strictly negative. The largest overall statistical error $\sigma'_\epsilon = 0.0117$ occurs for the bit string $\bar{x}_0\bar{x}_1 = 11$, whereas the largest absolute overall systematic error $|\mu'_\epsilon| = 0.1617$ occurs for the bit string $\bar{x}_0\bar{x}_1 = 10$. However, a discussion of possible physical origins of these hardware-induced errors goes beyond the scope of this manuscript.

6 Conclusions

We have proposed a quantum representation of classification trees in form of a quantum circuit, called Q-tree, which allows us to perform probabilistic tree traversals in the sense that it can sample from the corresponding probability distribution of the tree leaves. The proposed circuit consists of $d + k + m$ qubits and $\mathcal{O}(2^{k+m} + d2^d)$ gates (where d represents the tree depth, k the number of binary features and m the number of binary labels). Furthermore, we have also discussed how the sampled data can be used to feed a fitness function of a genetic algorithm to induce Q-trees in form of parameterized circuits. In addition, we have provided three approaches how Q-trees can be used for the prediction of labels for query data. First, by gathering samples and estimating the corresponding leaf probability distributions. And second, by modifying the circuit such that it directly draws samples from the distribution of leaves that a query distribution reaches, which allows to predict the labels for uncertain queries. As a third method, we have proposed an on-demand sampling approach to perform Q-tree predictions with a constant number of classical memory slots, independent of the tree depth. Using simple data sets, we have studied the induction and prediction capabilities of Q-trees in comparison with a classical benchmark using both a simulator and actual IBM quantum hardware. To our knowledge, this is the first realization of a decision tree classifier on a quantum device. Based on our experiments, we have found that the performance of Q-trees is comparable to the considered classical benchmark.

However, we have also found that the field of application of Q-trees is very limited on NISQ devices and hardware-induced noise can have a major impact on the quality of the results. Investigation of the hardware and data requirements necessary for the application of Q-trees could be the subject of future work. In addition, explicitly modifying the Q-trees to better fit the demands of the hardware could be one way to further improve the results and broaden the field of application. For example, we have mentioned how the circuit complexity can be reduced by introducing ancilla qubits (for both the data encoding and the structure encoding). An application of these approaches could lead to shallower Q-trees, which might be more suitable for NISQ devices. Furthermore, our proposed on-demand sampling approach can also be modified to mitigate hardware-induced noise

by employing the approach presented in Sec. 4.3. Further research is necessary to evaluate this topic in more detail.

There are various possibilities to extend or alter our proposed approach for different applications. For example, it can be generalized by considering decision trees with missing splits by an appropriate modification of the compressed decision configuration. Another obvious extension is the consideration of non-binary features and labels, respectively, which requires a suitable encoding with possibly more than one qubit per feature and label. Our approach could also be modified to allow decision trees for regression problems by employing a suitable probabilistic representation of the problem. All of these conceptional ideas can be considered as possible research directions for further work. Moreover, instead of single decision trees, another extension of our approach are ensembles of Q-trees in the sense of random forests [67]. In App. E, we briefly discuss a possible strategy for such an extension.

In this work, we have only discussed heuristic tree induction methods. We have specifically proposed a genetic algorithm in App. B, for which we have also suggested possible variations. However, quantum computers can in principle also be used to induce global optimal decision trees [5]. For this purpose, the corresponding mixed-integer optimization problem can for example be converted [68] into an unconstrained binary quadratic programming problem [69], which can then be solved with an appropriate quantum algorithm [70]. We consider such or a similar approach as another promising research direction for further studies.

Acknowledgements

We thank Michael Helmling, Michael Bortz, and Peter Klein for informative discussion and acknowledge the use of IBM Quantum services for this work. Part of this work has been funded by the Competence Center Quantum Computing Rhineland-Palatinate.

Authors' contributions

Raoul Heese developed the theoretical formalism, designed and conducted the experiments, and wrote the manuscript. Patricia Bickert contributed to the evaluation of the experiments and the writing. Astrid Elisa Niederle contributed to the overall research process.

A Compressed decision configuration

In this appendix section, we discuss the bijective transformation, Eq. (14), between the decision configuration E, Eq. (6), and the compressed decision configuration C, Eq. (12), where we implicitly assume that E fulfills Eq. (10). The intuition behind the compressed decision configuration is that each element c_j^i represents a swap of the elements of a k-dimensional index vector

$$\mathbf{J} \equiv (1,\dots,k) \in \{1,\dots,k\}^k \tag{91}$$

such that the actual feature index e_j^i corresponds to the $(i+1)$th element of this vector. We therefore call c_j^i a *swap index* and $\mathbf{c}^i$ the vector of *layer swaps*. For the root, there exists a direct correspondence

$$c_1^0 = e_1^0 \tag{92}$$

between the two parameterizations.

Furthermore, for any depth $l > 0$, one has the transformation rule

$$c^l_{\nu_l(\bar{x}_0\cdots\bar{x}_{l-1})} \equiv w(\mathbf{T}^0(l-1;\bar{x}_0\cdots\bar{x}_{l-1}), e^l_{\nu_l(\bar{x}_0\cdots\bar{x}_{l-1})}) \tag{93}$$

for $E \mapsto C$, where we denote the index of a value $n \in \{1,\dots,k\}$ in a vector $\mathbf{I} \in \{1,\dots,k\}^k$ with

$$w(\mathbf{I}, n) \equiv \operatorname*{arg\,max}_{i\in\{1,\dots,k\}} |I_i - n| \tag{94}$$

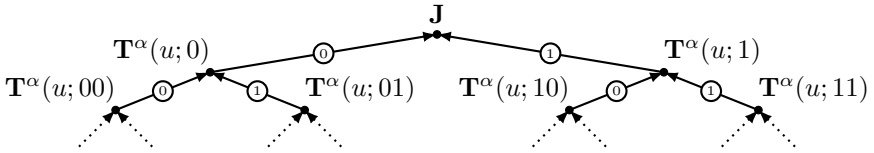


Figure 18: Illustration of the transformations $\mathbf{T}^\alpha(u;\bar{x}_0\cdots\bar{x}_{l-1})$, Eq. (95), of the index vector $\mathbf{J}$, Eq. (91), by mapping them to the corresponding nodes of a decision tree T, cf. Fig. 1. In each layer, a permutation of the vector elements according to Eq. (97) is performed (with the arrows indicating dependencies on previous permutations). The compressed decision configuration C, Eq. (12), describes the corresponding index swaps in form of a sequence of mutually independent swap indices c^i_j, Eq. (13), with $j\in\{1,\ldots,2^i\}$. In contrast, the decision configuration E, Eq. (6), directly describes the feature indices of splits in form of mutually dependent layer decisions e^i_j, Eq. (7), which have to fulfill Eq. (10). The two representations can be converted into each other, Eq. (14), according to Eqs. (92), (93) and (100).

and require that n exists exactly once in $\mathbf{I}$. Moreover, we make use of the vector transformation

$$\mathbf{T}^\omega(u;\bar{x}_0\cdots\bar{x}_{l-1}) \equiv \left(\mathop{\mathrm{C}}_{v=0}^{u}\mathbf{V}^\omega(\cdot,v;\bar{x}_0\cdots\bar{x}_{l-1})\right)(\mathbf{J}) \tag{95}$$

with $\omega\in\{0,1\}$ and $u\in\{l-1,l\}$. It contains the composition

$$\mathop{\mathrm{C}}_{v=0}^{u}\mathbf{V}^\omega(\cdot,v) \equiv \mathbf{V}^\omega(\cdot,u)\circ\cdots\circ\mathbf{V}^\omega(\cdot,0) \tag{96}$$

of

$$\mathbf{V}^\omega(\mathbf{I},v) \equiv \mathbf{V}^\omega(\mathbf{I},v;\bar{x}_0\cdots\bar{x}_{l-1}) \equiv \begin{cases}\mathbf{S}(\mathbf{I},v+1,w(\mathbf{I},e^v_{\nu'_v(\bar{x}_0\cdots\bar{x}_{l-1})})) & \text{for } \omega=0\\ \mathbf{S}(\mathbf{I},v+1,c^v_{\nu'_v(\bar{x}_0\cdots\bar{x}_{l-1})}) & \text{for } \omega=1\end{cases} \tag{97}$$

with $0\leq v\leq l$. The latter is based on the permutation (element swap)

$$\mathbf{S}(\mathbf{I},l,n) = \begin{cases}(I_1,\ldots,I_{l-1},I_n,I_{l+1},\ldots,I_{n-1},I_l,I_{n+1},\ldots,I_k) & \text{for } l<n\\ \mathbf{I} & \text{for } n=l\\ (I_1,\ldots,I_{n-1},I_l,I_{n+1},\ldots,I_{l-1},I_n,I_{l+1},\ldots,I_k) & \text{for } l>n\end{cases} \tag{98}$$

of $\mathbf{I}$ with $l,n\in\{1,\ldots,k\}$. We also recall Eq. (8) and introduce

$$\nu'_v(\bar{x}_0\cdots\bar{x}_{l-1}) \equiv \begin{cases}\nu_v(\bar{x}_0\cdots\bar{x}_{v-1}) & \text{for } v\geq 1\\ 1 & \text{for } v=0\end{cases} \tag{99}$$

with $0\leq v\leq l$ to identify the nodes along the subpath (with depth $v<l$) of a node of depth l, which is identified by a bit string $\bar{x}_0\cdots\bar{x}_{l-1}$. In particular, one has $\nu'_l(\bar{x}_0\cdots\bar{x}_{l-1})=\nu_l(\bar{x}_0\cdots\bar{x}_{l-1})$.

Conversely, the inverse transformation rule $C\mapsto E$ for any depth $l>0$ reads

$$e^l_{\nu_l(\bar{x}_0\cdots\bar{x}_{l-1})} = T^1_{l+1}(l;\bar{x}_0\cdots\bar{x}_{l-1}) \tag{100}$$

such that the feature indices e^i_j are defined by the corresponding elements of Eq. (95). These elements represent a permutation of the elements of the index vector $\mathbf{J}$, Eq. (91). We briefly illustrate the transformations of $\mathbf{J}$ in Fig. 18.

Summarized, a decision tree T can be parameterized by either E, Eqs. (6) and (10), or C, Eq. (12). The two representations can be converted into each other, Eq. (14), according to Eqs. (92), (93) and (100).

B Genetic induction algorithm

In the present appendix section, we describe a concrete realization of a genetic algorithm for the induction of a decision tree T, Eq. (15), as introduced in Sec. 3.2. For this purpose, we assume

Algorithm 1 Genetic tree induction algorithm

function GeneticGrow(θ)
  $\chi^{\text{best}} \leftarrow$ Initalize(1)
  if $P > 1$ **then**
    $\chi \leftarrow$ Initalize(P)
    $g \leftarrow 0$
    while $g < G$ **do**
      $\chi_1^{\text{best}} \leftarrow$ Best($\chi, f_{\text{ent}}, f_{\text{bac}}$)
      $\chi^{\text{gen}} \leftarrow$ Select($\chi, t, P-1, f_{\text{ent}}, f_{\text{bac}}$)
      for all $u \in \{2j \,|\, j \in \mathbb{N} \wedge 2 \leq 2j \leq P-1\}$ **do**
        for all $v \in \{2j-1 \,|\, j \in \mathbb{N} \wedge 1 \leq 2j-1 \leq P-1\}$ **do**
          $\chi_u^{\text{gen}}, \chi_v^{\text{gen}} \leftarrow$ Crossover($\chi_u^{\text{gen}}, \chi_v^{\text{gen}}, p_{\text{c}}$)
        end for
      end for
      for all $u \in \{1, \ldots, P-1\}$ **do**
        $\chi_u^{\text{gen}} \leftarrow$ Mutate($\chi_u^{\text{gen}}, p_{\text{m}}, p_{\text{a}}$)
      end for
      $\chi \leftarrow \chi^{\text{gen}} \cup \chi^{\text{best}}$
      $g \leftarrow g + 1$
    end while
  else
    $\chi \leftarrow \chi^{\text{best}}$
  end if
  return Best($\chi, f_{\text{ent}}, f_{\text{bac}}$)
end function

that a constant maximum depth d is given for the tree to be induced. Furthermore, one can either choose to use the classical or the quantum representation of the tree T as described in Secs. 3 and 4, respectively.

The proposed genetic algorithm follows the typically used "traditional" structure and makes use of well-known genetic operators [21]. It is outlined in Alg. 1 and particularly depends on the hyperparameters

$$\theta \equiv (P, G, t, p_{\text{c}}, p_{\text{m}}, p_{\text{a}}, f_{\text{ent}}, f_{\text{bac}}) \tag{101}$$

consisting of the total population size $P \in \mathbb{N}$, the number of iterations (or generations) $G \in \mathbb{N}$, the selection subset size $t \in \mathbb{N}$, the probability of mating two individuals $p_{\text{c}} \in [0,1]$, the probability of mutating an individual $p_{\text{m}} \in [0,1]$, the probability for each attribute to be mutated independently $p_{\text{a}} \in [0,1]$, and the fitness function weights $f_{\text{ent}}, f_{\text{bac}} \in \mathbb{R}_{0+}$. We denote a population of $P' \in \mathbb{N}$ chromosomes (i. e., candidate solutions) of the form defined in Eq. (16) by the P'-tuple $\chi \in \mathcal{S}^{P'}$, where we recall the solution domain $\mathcal{S}$, Eq. (17). The chosen maximum depth d consequently defines the dimensionality of the chromosomes. Furthermore, we denote the ith chromosome by $\chi_i^{P'} \in \mathcal{S}$ with $i \in \{1, \ldots, P'\}$.

A central component of the algorithm is the fitness function, which we define as

$$F_{\text{cl/qm}} \equiv F_{\text{cl/qm}}(f_{\text{ent}}, f_{\text{bac}}) \equiv f_{\text{ent}} F_{\text{cl/qm}}^{\text{ent}} + f_{\text{bac}} F_{\text{cl/qm}}^{\text{bac}}. \tag{102}$$

It consists of two parts weighted by f_{ent} and f_{bac}, respectively. The first part

$$F_{\text{cl/qm}}^{\text{ent}} \equiv F_{\text{cl/qm}}^{\text{ent}}[p_{\text{cl/qm}}(\mathcal{C}_\nu), p_{\text{cl/qm}}(y_i \,|\, \mathcal{C}_\nu)] \equiv \frac{-1}{m} \sum_{\nu=1}^{2^d} p_{\text{cl/qm}}(\mathcal{C}_\nu) \sum_{i=1}^{m} S[p_{\text{cl/qm}}(y_i \,|\, \mathcal{C}_\nu)] \tag{103}$$

is a measure for the average entropy of the probability distributions of the leaves of the tree, where the first sum iterates over the indices ν of all leaves of the tree T, Eq. (8), with the corresponding set of conditions $\mathcal{C}_\nu$, Eq. (28). Here we introduce the information entropy [71]

$$S[p(y)] \equiv -p(y=0) \log_2 p(y=0) - p(y=1) \log_2 p(y=1) \tag{104}$$

of a probability distribution $p(y)$ with support $\mathbb{B}$. The second part

$$F_{\mathrm{cl/qm}}^{\mathrm{bac}} \equiv F_{\mathrm{cl/qm}}^{\mathrm{bac}}(\mathbf{D}_{\mathrm{train}}) \equiv \frac{1}{2m} \sum_{i=1}^{m} \sum_{b=0}^{1} \frac{\mathrm{t}_{\mathrm{cl/qm}}^{b}(i, \mathbf{D}_{\mathrm{train}})}{\mathrm{t}_{\mathrm{cl/qm}}^{b}(i, \mathbf{D}_{\mathrm{train}}) + \mathrm{f}_{\mathrm{cl/qm}}^{b}(i, \mathbf{D}_{\mathrm{train}})} \tag{105}$$

represents the mean balanced accuracy in accordance with Eq. (78). Here we make use of the abbreviations

$$\mathrm{t}_{\mathrm{cl/qm}}^{b}(i, \mathbf{D}_{\mathrm{train}}) \equiv |\{\mathbf{d} \,|\, (\mathbf{x}^q, \mathbf{y}) = \mathbf{d} \in \mathrm{t}_{\mathrm{cl/qm}}(\mathbf{D}_{\mathrm{train}}) \wedge y_i = b\}| \tag{106}$$

and

$$\mathrm{f}_{\mathrm{cl/qm}}^{b}(i, \mathbf{D}_{\mathrm{train}}) \equiv |\{\mathbf{d} \,|\, (\mathbf{x}^q, \mathbf{y}) = \mathbf{d} \in \mathrm{f}_{\mathrm{cl/qm}}(\mathbf{D}_{\mathrm{train}}) \wedge y_i = b\}|, \tag{107}$$

which are based on

$$\mathrm{t}_{\mathrm{cl/qm}}(i, \mathbf{D}_{\mathrm{train}}) \equiv |\{\mathbf{d} \,|\, (\mathbf{x}^q, \mathbf{y}) = \mathbf{d} \in \mathbf{D}_{\mathrm{train}} \wedge y_i = \hat{y}_{\mathrm{cl/qm}}(i, \mathbf{x}^q)\}| \tag{108}$$

and

$$\mathrm{f}_{\mathrm{cl/qm}}(i, \mathbf{D}_{\mathrm{train}}) \equiv |\{\mathbf{d} \,|\, (\mathbf{x}^q, \mathbf{y}) = \mathbf{d} \in \mathbf{D}_{\mathrm{train}} \wedge y_i \neq \hat{y}_{\mathrm{cl/qm}}(i, \mathbf{x}^q)\}|, \tag{109}$$

respectively, where $b \in \mathbb{B}$ and $i \in \{1, \ldots, m\}$.

Depending on whether the classical or the quantum representation of the tree T is considered, one has

$$F_{\mathrm{cl/qm}} \equiv \begin{cases} F(\mathbf{C}, \mathbf{D}_{\mathrm{train}}) & \text{for the classical representation} \\ \hat{F}(b_1, \ldots, b_N, \mathbf{D}_{\mathrm{train}}) & \text{for the quantum representation} \end{cases} \tag{110}$$

with the exact classical fitness function $F(\mathbf{C}, \mathbf{D}_{\mathrm{train}})$, Eq. (18), and the measured fitness function from the quantum representation $\hat{F}(b_1, \ldots, b_N, \mathbf{D}_{\mathrm{train}})$, Eq. (62), respectively. Furthermore, one has

$$p_{\mathrm{cl/qm}}(\mathcal{C}_\nu) \equiv \begin{cases} p(\mathcal{C}_\nu) & \text{for the classical representation} \\ \hat{p}(\mathcal{C}_\nu) & \text{for the quantum representation} \end{cases} \tag{111}$$

for the probability distribution of reaching a certain node, where we recall Eqs. (30) and (60), and

$$p_{\mathrm{cl/qm}}(y_i \,|\, \mathcal{C}_\nu) \equiv \begin{cases} p(y_i \,|\, \mathcal{C}_\nu) & \text{for the classical representation} \\ \hat{p}(y_i \,|\, \mathcal{C}_\nu) & \text{for the quantum representation} \end{cases} \tag{112}$$

for the probability distribution of the ith label with $i \in \{1, \ldots, m\}$, where we recall Eqs. (32) and (61), respectively. Finally, one has

$$\hat{y}_{\mathrm{cl/qm}}(i, \mathbf{x}^q) \equiv \arg\max_{y_i} \begin{cases} p(y_i \,|\, \mathcal{C}_{q_d(\mathbf{x}^q)}) & \text{for the classical representation} \\ \hat{p}_{\mathrm{cl}}(y_i \,|\, \mathbf{x}^q) & \text{for the quantum representation} \end{cases} \tag{113}$$

for the prediction of the ith label according to Eqs. (40) and (64), respectively.

The information entropy, Eq. (104), attains its smallest value 0 for $p(y = 0) \in \{0, 1\}$ and its largest value 1 for $p(y = 0) = \frac{1}{2}$. The choice $p(y) = p_{\mathrm{cl/qm}}(y_i \,|\, \mathcal{C}_\nu)$ therefore leads to the following implication: Since a smaller information entropy corresponds to a less equal partitioning of data with different labels (in the sense of a smaller variance), it indicates a more favorable splitting decision with respect to its data partitioning capabilities.

In particular, the information entropy of two simultaneous events is no more than the sum of the information entropies of each individual event [72]. A splitting decision can therefore not increase the information entropy such that

$$S[p(y_i)] \geq \sum_{\nu=1}^{2^d} p(\mathcal{C}_\nu) S[p(y_i|\mathcal{C}_\nu)] \tag{114}$$

holds true.

This measure is in particular closely related to the well-known information gain criterion [2] for the induction of decision trees. Maximization of $F^{\text{ent}}_{\text{cl/qm}}$ is equivalent to finding the minimum weighted average of the information entropy in all leaves over all label dimensions and can therefore be considered as a measure for the partitioning capabilities of all splitting decisions in the tree. An increased data partitioning is favorable since it represents a more conclusive tree.

On the other hand, the balanced accuracy, Eq. (105), is a measure of how well the splitting decisions are able to explain the training data $\mathbf{D}_{\text{train}}$. In the fitness function, we combine both the conclusiveness Eq. (103) and the accuracy Eq. (105) to a combined metric for the quality of the splitting decisions in the tree (defined by $\mathbf{C}$ and $\mathbf{D}_{\text{train}}$ or $b_1, \dots, b_N$). By tuning the (non-negative) weights f_{ent} and f_{bac}, the influence of the two parts can be controlled for a specific use case. For the sake of convenience, one can also set $f_{\text{bac}} = 1 - f_{\text{ent}}$ to eliminate one hyperparameter.

Alg. 1 contains the following functions:

- INITALIZE(P'): P' chromosomes are created by drawing them uniformly from an appropriate set of possible values, Eq. (17). The chromosomes are returned as a P'-tuple, where $P' \in \{1, P\}$.
- BEST($\chi, f_{\text{ent}}, f_{\text{bac}}$): Return the chromosome in χ with the best fitness, Eq. (102).
- SELECT($\chi, t, P-1, f_{\text{ent}}, f_{\text{bac}}$): Selection of $P-1$ chromosomes with the *tournament selection* method. Specifically, t chromosomes are drawn uniformly from χ and the one with the best fitness, Eq. (102), is selected. This process is repeated $P-1$ times and the $(P-1)$-tuple of selected chromosomes χ^{gen} is returned.
- CROSSOVER($\chi^{\text{gen}}_u, \chi^{\text{gen}}_v, p_{\text{c}}$): Uniformly draw a value r from $[0, 1]$ and perform a *single-point crossover* if $r \leq p_{\text{c}}$, otherwise return the unmodified originals χ^{gen}_u and χ^{gen}_v. The single-point crossover begins with uniformly drawing an index i of the chromosome attributes from $\{2, \dots, 2^d - 1\}$ at which χ^{gen}_u and χ^{gen}_v are each splitted into two segments. Then, two new chromosomes are created. The first corresponds to a concatenation of the first segment of χ^{gen}_u and the second segment of χ^{gen}_v, whereas the second corresponds to a concatenation of the first segment of χ^{gen}_v and the second segment of χ^{gen}_u, respectively. These two copies are returned.
- MUTATE($\chi^{\text{gen}}_u, p_{\text{m}}, p_{\text{a}}$): Uniformly draw a value r from $[0, 1]$ and perform a *uniform mutation* if $r \leq p_{\text{m}}$, otherwise return the unmodified original χ^{gen}_u. The uniform mutation begins with uniformly drawing a value r_i from $[0, 1]$ for each index $i \in \{1, \dots, 2^d\}$ of the chromosome χ^{gen}_u. Subsequently, it is checked whether $r_i \leq p_{\text{a}}$ is fulfilled for all indices and if yes, the corresponding attribute of the chromosome is replaced by a uniformly drawn value from the set of possible values, Eq. (17). The mutated chromosome is returned.

By keeping track of the fittest chromosome in $\chi^{\text{best}} \in \mathcal{S}^1$, we ensure that best fitness in each generation cannot be worsened. This approach is also known as *elitism* [73]. The returned result of GENETICGROW(θ) is a chromosome that maximizes $F_{\text{cl/qm}}$ in the sense of Eq. (19).

The evaluation of the fitness function $F_{\text{cl/qm}}$ in SELECT($\chi, t, P-1, f_{\text{ent}}, f_{\text{bac}}$) and BEST($\chi, f_{\text{ent}}, f_{\text{bac}}$), respectively, implicitly depends on $\mathbf{D}_{\text{train}}$ and possibly $b_1, \dots, b_N$, depending on whether the classical or the quantum representation of the tree T is considered, Eq. (110). In particular, $\hat{F}(b_1, \dots, b_N, \mathbf{D}_{\text{train}})$ represents a noisy fitness function. To handle this case, we perform an evaluation of $\hat{F}(b_1, \dots, b_N, \mathbf{D}_{\text{train}})$ and assign the fitness to the corresponding chromosome on its initialization or whenever it is affected by a crossover or a mutation. In addition, we average the fitness values over identical chromosomes in χ at the end of each iteration (i. e., after line 18) and re-assign the mean fitness to the associated chromosomes. By doing so we can reduce the influence of statistical errors on the final outcome. The on-demand sampling proposed in Sec. 4.3 can also be used to reduce the classical memory required for the evaluation of the quantum fitness function.

The algorithm presented here can be varied in many different ways, which may lead to better or worse results depending on the specific application at hand. For example, instead of having a constant maximum depth d, a variable maximum depth can be achieved by considering chromosomes of different length, which can each be assigned up to a specified maximum depth. Furthermore,

instead of optimizing the tree as a whole, an iterative layer-by-layer optimization could be performed by optimizing only the decision configuration of the current layer from depth 0 to depth $d-1$ (or up to a variable depth). Finally, the fitness function $F_{\mathrm{cl/qm}}$ can be modified in various ways, for example by incorporating another common splitting criterion like the Gini index [2]. One could also consider a multi-objective genetic algorithm [21] with one fitness value for each of the two parts or one fitness value for each label (instead of performing the average over all labels $i \in \{1,\dots,m\}$ in Eqs. (103) and (105)).

C Q-tree measurements

In Secs. 4.1.1 and 4.1.2, we introduce the Q-tree data encoding and structure encoding, respectively. In the present section, we briefly show how the form of these encodings can be used to derive the probability distribution of the subsequent projective measurement, Eq. (56), which is important for the discussion in Sec. 4.1.2.

We begin by recognizing that $\bar{x}_0 \cdots \bar{x}_{i-1} = \kappa(j,i,1)\cdots\kappa(j,i,i)$ in fact represents a path within T. If the corresponding set of conditions $x_q = \kappa(j,i,u)\,\forall\, u \in \{1,\dots,i\}$ is fulfilled, then $a_\Gamma(x_1,\dots,x_k) = a(x_{V_1},\dots,x_{V_k})$, where we recall $\mathbf{V} \equiv \mathbf{V}^1(\mathbf{I}, i; \kappa(j,i,1)\cdots\kappa(j,i,i))$, Eq. (97).

Thus,

$$|\Psi''(T)\rangle = \sum_{\mathbf{x},\mathbf{y}} \sqrt{p_T(\mathbf{x},\mathbf{y})}\,|\mathbf{x},\mathbf{y}\rangle \tag{115}$$

according to Eq. (43). Here, we introduce

$$p_T(\mathbf{x},\mathbf{y}) \equiv \begin{cases} p(x_{T_1(1)},\dots,x_{T_k(1)},\mathbf{y}) & \text{for } x_q = \kappa(1,d,u)\,\forall\, u \in \{1,\dots,d\} \\ \vdots & \vdots \\ p(x_{T_1(2^d)},\dots,x_{T_k(2^d)},\mathbf{y}) & \text{for } x_q = \kappa(2^d,d,u)\,\forall\, u \in \{1,\dots,d\} \\ p(\mathbf{x},\mathbf{y}) & \text{otherwise} \end{cases} \tag{116}$$

and recall the vector transformation $\mathbf{T}(j) \equiv \mathbf{T}^1(d;\kappa(j,d,1)\cdots\kappa(j,d,d))$, Eq. (95) for all paths $j \in \{1,\dots,2^d\}$. From Eqs. (115) and (116), we can straightforwardly infer Eq. (56).

D Uncertainty

In the present appendix section, we summarize how we evaluate the statistical and hardware-induced uncertainties of probabilities estimated from fractions of measurement counts [74] using frequentist confidence intervals [75]. First, we consider the ratio of a binomial random variable to a constant number of trials, which allows us to determine confidence intervals for Eqs. (60) and (69), respectively. Second, to determine confidence intervals of Eqs. (61) and (63), we consider the case where the number of trials is also a binomial random variable. Finally, we use error propagation to estimate the uncertainty of Eq. (70). We conclude with a brief discussion of hardware-induced noise. A mathematical discussion of quantum sampling and its relation to classical sampling can be found, e. g., in [76].

First, presume a random variable $n \sim B(n;N,p)$ following the binomial distribution

$$B(n;N,p) \equiv \binom{N}{n} p^n (1-p)^{N-n} \tag{117}$$

with $N \in \mathbb{N}$ independent Bernoulli trials of success probability $p \in [0,1]$ such that $n \in \{0,\dots,N\}$ represents the drawn number of successes. Since the expectation value of the fraction of successes obeys

$$\mathbb{E}\left[\frac{n}{N}\right] = p, \tag{118}$$

the success probability for a finite set of trials can be estimated by the fraction of successes according to

$$p \approx \hat{p}(n,N) \equiv \frac{n}{N} \tag{119}$$

with standard deviation

$$\sigma\hat{p}(p,N) = \sqrt{\frac{p(1-p)}{N}} \approx \hat{\sigma}\hat{p}(n,N) \equiv \sqrt{\frac{\hat{p}(n,N)(1-\hat{p}(n,N))}{N}}, \tag{120}$$

which we use as a confidence interval [77]

$$\hat{p}(n,N) \pm z_{\alpha/2}\hat{\sigma}\hat{p}(n,N) \tag{121}$$

for p with target error rate $\alpha \in [0,1]$. Here, $z_{\alpha/2}$ denotes the $1-\alpha/2$ quantile of the standard normal distribution. In particular, Eq. (121) can be used to estimate the confidence intervals of Eqs. (60) and (69), respectively. For this purpose, we set $n \equiv n(\mathcal{C}_\nu)$ and identify N with the total number of measurements.

Second, we consider the estimation of a fraction k/m of two random variables $k, m \sim D(k,m;N,p',q)$, where the the probability distribution of jointly drawing k and m is given by the product

$$D(k,m;N,p',q) \equiv B(k;m,p')B_1(m;N,q) \tag{122}$$

with $p' \in [0,1]$, $q \in (0,1]$ and $N \in \mathbb{N}$ such that $k \in \{0,\ldots,m\}$ and $m \in \{1,\ldots,N\}$, respectively. Here, we recall the binomial distribution, Eq. (117), and introduce the truncated binomial distribution

$$B_1(m;N,q) \equiv \frac{B(m;N,q)}{b_0(N,q)}, \tag{123}$$

which considers only one or more successes (i. e., $m \geq 1$) and contains the abbreviation

$$b_0(N,q) \equiv 1 - B(0;N,q) = 1-(1-q)^N \in (0,1] \tag{124}$$

with $b_0(N,q) \approx 1$ for $N \gg 1$. One has

$$\mathbb{E}\left[\frac{k}{m}\right] = p' \tag{125}$$

and

$$\mathbb{E}\left[\frac{m}{N}\right] = \frac{q}{b_0(N,q)}, \tag{126}$$

respectively, in analogy to Eq. (118) and consequently

$$p' \approx \hat{p}'(k,m) \equiv \frac{k}{m} \tag{127}$$

in analogy to Eq. (119). We use the corresponding standard deviation

$$\sigma\hat{p}'(N,p',q) = \sqrt{p'(1-p')\,\mathbb{E}\left[\frac{1}{m}\right]} \approx \hat{\sigma}\hat{p}'(k,m,N) \equiv \sqrt{\hat{p}'(k,m)(1-\hat{p}'(k,m))\zeta(N,m/N)} \tag{128}$$

with the expectation value

$$\mathbb{E}\left[\frac{1}{m}\right] = \frac{Nq(1-q)^{N-1}{}_4F_2(\{1,1,1-N\},\{2,2\},\frac{q}{q-1})}{b_0(N,q)} \equiv \zeta(N,q) \tag{129}$$

as a confidence interval

$$\hat{p}'(k,m) \pm z_{\alpha/2}\hat{\sigma}\hat{p}'(k,m,N) \tag{130}$$

in analogy to Eq. (121). Here,

$$ {}_rF_s(\{a_1,\dots,a_r\},\{b_1,\dots,b_s\},z) \equiv \sum_{l=0}^{\infty} \frac{(a_1)_l \cdots (a_r)_l}{(b_1)_l \cdots (b_s)_l} \frac{z^l}{l!} \tag{131} $$

denotes the generalized hypergeometric function with the Pochhammer symbol $(\cdot)_l$. Furthermore, we use $q \approx m/N$ according to Eq. (126) with $N \gg 1$. Eq. (130) can be used to estimate the confidence interval of Eqs. (61) and (63). For this purpose, we set $k \equiv n_i^{y_i}(\mathcal{C}_\nu)$, $m \equiv n(\mathcal{C}_\nu)$, and identify N with the total number of measurements.

Finally, we determine the uncertainty of Eq. (70) using standard error propagation (i. e., we neglect correlations and assume independent variables), which yields the standard deviation

$$ \sigma\hat{p}_{\mathrm{qm}}(y_i \,|\, p^q) \equiv \sqrt{\sum_{\nu=1}^{2^d} \Big[\hat{p}(y_i \,|\, \mathcal{C}_\nu)^2 \, \sigma\hat{p}^q(\mathcal{C}_\nu)^2 + \hat{p}^q(\mathcal{C}_\nu)^2 \, \sigma\hat{p}(y_i \,|\, \mathcal{C}_\nu)^2\Big]}. \tag{132} $$

Here, $\sigma\hat{p}^q(\mathcal{C}_\nu)$ and $\sigma\hat{p}(y_i \,|\, \mathcal{C}_\nu)$ represent the standard deviations of the corresponding probability densities, Eqs. (61) and (69), which are given by Eqs. (120) and (128), respectively.

So far, we have only discussed statistical uncertainties. However, measurements on real quantum computers may also exhibit additional uncertainties from hardware imperfections. Such noise-induced uncertainties may, e. g., occur for Eqs. (63), (69) and (70). In general, noise-free measurement results from quantum computers are obtained by sampling from a probability distribution

$$ P(\rho, \hat{M}) = \mathrm{Tr}\Big\{\hat{M}^\dagger \hat{M} \rho\Big\} \tag{133} $$

based on a density matrix ρ, which results from the application of gate operations on the initial state, and a measurement operator $\hat{M}$ (with Hermitian conjugate $\hat{M}^\dagger$). According to Eq. (56), the probabilities p, p', and q from Eqs. (117) and (122) can be identified with an expression of the form of Eq. (133) when applied to Eqs. (63), (69) and (70), respectively.

When including noise from hardware imperfections [78], one can use a master equation approach [79] to obtain

$$ P(\rho_\epsilon, \hat{M}) = \mathrm{Tr}\Big\{\hat{M}^\dagger \hat{M} \rho_\epsilon\Big\} \tag{134} $$

with the noise-perturbed density matrix ρ_ϵ, which can be expressed by [80]

$$ \rho_\epsilon \approx \rho + \delta\rho \tag{135} $$

based on the approximated noise perturbation $\delta\rho$ such that

$$ P(\rho_\epsilon, \hat{M}) \approx P(\rho, \hat{M}) + \mathrm{Tr}\Big\{\hat{M}^\dagger \hat{M} \delta\rho\Big\}. \tag{136} $$

This perturbation of the probability can also be included in Eqs. (117) and (122) if we treat the probabilities p, p', and q as random variables with a probability distribution determined by the hardware noise. Based on this presumption, we consider the generalizations

$$ B_\epsilon(n; N, p) \equiv \int_0^1 \epsilon(\tilde{p}; p) B(n; N, \tilde{p}) \, \mathrm{d}\tilde{p} \tag{137} $$

and

$$ D_\epsilon(k, m; N, p', q) \equiv \int_0^1 \int_0^1 \epsilon(\tilde{p}, \tilde{q}; p', q) D(k, m; N, \tilde{p}, \tilde{q}) \, \mathrm{d}\tilde{p} \, \mathrm{d}\tilde{q}, \tag{138} $$

respectively. Here we introduce the probability density $\epsilon(\tilde{p}; p)$ with support $[0, 1]$ and the probability density $\epsilon(\tilde{p}, \tilde{q}; p', q)$ with support $[0, 1]^2$ to quantify the hardware-induced uncertainty.

Based on these presumptions, suitably chosen probability densities for p, p', and q can be used to model the noise (e. g., a truncated normal distributions for a heuristic approach or a

distribution derived from the explicit form of $P(\rho_\epsilon, \hat{M})$ for a rigorous approach). Such a noise model may lead to a shift of expectation values, Eqs. (118) and (125), which corresponds to hardware-induced systematic errors, and an increase of the standard deviations, Eqs. (120) and (128), which corresponds to a hardware-induced uncertainty.

In the noise-free case, these probability densities can be expressed in terms of Dirac delta distributions

$$\epsilon(\tilde{p}; p) = \delta(\tilde{p} - p) \tag{139}$$

and

$$\epsilon(\tilde{p}, \tilde{q}; p', q) = \delta(\tilde{p} - p')\delta(\tilde{q} - q), \tag{140}$$

respectively, such that Eqs. (137) and (138) reduce to Eqs. (117) and (122) and the statistical uncertainty becomes decisive. On the other hand, since the standard deviations from Eqs. (120) and (128) decrease for an increasing number of trials N, we can write

$$B(n; N \gg 1, p) \approx \delta\left(\frac{n}{N} - p\right) \tag{141}$$

and

$$D(k, m; N \gg 1, p', q) \approx \delta\left(\frac{k}{m} - p'\right)\delta\left(\frac{m}{N} - q'\right) \tag{142}$$

for a sufficiently large N. Hence, Eqs. (137) and (138) reduce to

$$B_\epsilon(n; N, p) \approx \epsilon\left(\frac{n}{N}; p\right) \tag{143}$$

and

$$D_\epsilon(k, m; N, p', q) \approx \epsilon\left(\frac{k}{m}, \frac{m}{N}; p', q\right), \tag{144}$$

respectively. This means that in this case the statistical uncertainty is negligible and the hardware-induced uncertainty (if present) is dominating.

E Random forests

Random forests represent a well-known predictive model composed of an ensemble of decision trees for which a prediction can be inferred from the (weighted) averaged predictions of the individual trees [67]. Our proposed quantum representation of binary classification trees can be generalized to such ensembles using additional qubits and gates. The connection between decision tree ensembles and quantum computation has already been explored in previous works. For example, a conceptional quantum forest model based on quantum amplitude amplification is presented in [81]. A general discussion of quantum ensembles of quantum classifiers can be found in [82]. Furthermore, a quantum-inspired (i. e., involving no actual quantum computation) feature subset generation method for ensemble methods is presented in [83]. Furthermore, a discussion of probabilistic random forests can be found in [28].

An ensemble of $f \in \mathbb{N}$ trees $T(1), \ldots, T(f)$ of the same maximum depth d can by characterized by the training data $\mathbf{D}_{\text{train}}$, Eq. (5), and the f-tuple

$$F \equiv (C(1), \ldots, C(f)), \tag{145}$$

which consists of the compressed decision configurations of all trees, Eq. (12). For the tth tree $T(t)$, the compressed decision configuration is denoted by $C(t)$ with $t \in \{1, \ldots, f\}$. An additional probability distribution p_{trees} with support $\{1, \ldots, f\}$ represents the relative importance of every tree in the forest by which its prediction is weighted.

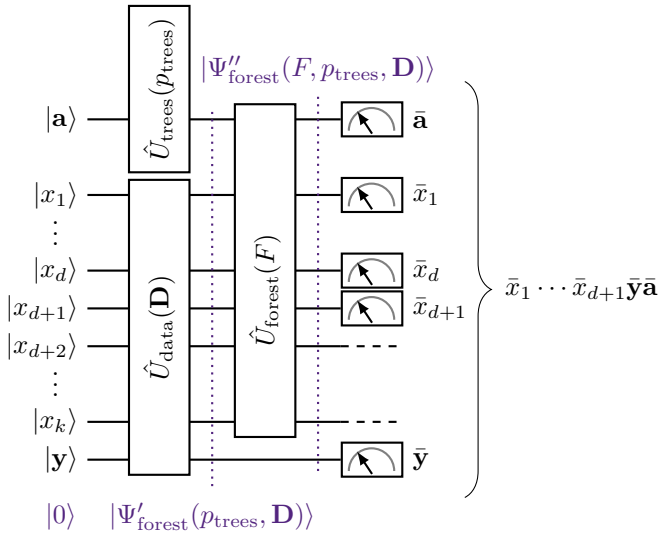


Figure 19: Q-Forest (circuit). Layout for our proposed quantum representation of a random forest consisting of f trees $T(1), \ldots, T(f)$ of maximum depth d, which is uniquely defined by the training data $\mathbf{D}_{\text{train}}$, the tuple of compressed decision configurations F, Eq. (145), and the probability distribution of the trees p_{trees}. The Q-Forest is a generalization of the Q-tree, Fig. 3. Initially, all qubits are prepared in the ground state $|0\rangle$. The first set of unitary operators $\hat{U}_{\text{data}}(\mathbf{D}_{\text{train}})$ and $\hat{U}_{\text{trees}}(p_{\text{trees}})$ perform an encoding of the training data and decision tree distribution, respectively, that leads to the state $|\Psi'_{\text{forest}}(p_{\text{trees}}, \mathbf{D}_{\text{train}})\rangle$, Eq. (148). Second, $\hat{U}_{\text{forest}}(F)\,|\Psi'_{\text{forest}}(p_{\text{trees}}, \mathbf{D}_{\text{train}})\rangle$ performs an encoding of the decision configuration that leads to the state $|\Psi''_{\text{forest}}(F, p_{\text{trees}}, \mathbf{D}_{\text{train}})\rangle$, Eq. (149). Third, a projective measurement on is performed to obtain a bit string, Eq. (155), consisting of features, labels, and ancillas, which are drawn from Eq. (156) such that the circuit can be considered a quantum representation of the random forest.

A quantum representation of such a forest can be realized in analogy to our Q-tree, Sec. 4.1. In addition to the $k + m$ qubits that represent the features and labels, the *Q-forest circuit*, which we also refer to as *Q-forest* for short, requires

$$f' \equiv \lceil \log_2 f \rceil \tag{146}$$

additional ancilla qubits (denoted by $|a_1\rangle, \ldots, |a_{f'}\rangle$ or their respective indices $k + m + 1, \ldots, k + m + f'$), which identify each tree. For the sake of simplicity, we assume that f is chosen such that $f \geq 2$ and $\log_2 f \in \mathbb{N}$. We assume that all qubits are initially prepared in the ground state $|0\rangle$ in analogy to Eq. (41). The circuit consists of three parts. First, two unitary operators responsible for the data encoding. Second, a unitary operator responsible for the structural encoding and, third, a projective measurement. The proposed Q-forest circuit layout is sketched in Fig. 19. In the first step, a qsample encoding of the training data is performed by means of $\hat{U}_{\text{data}}(\mathbf{D}_{\text{train}})$ to obtain $|\Psi'(F, \mathbf{D}_{\text{train}})\rangle$, Eq. (42). In analogy, the distribution of trees p_{trees} is stored with a qsample encoding by means of $\hat{U}_{\text{trees}}(p_{\text{trees}})$, which only acts on the ancilla qubits such that

$$\hat{U}_{\text{trees}}(p_{\text{trees}})\,|0\rangle = \sum_{\mathbf{a}} \sqrt{p_{\text{trees}}(\mathbf{a})}\,|\mathbf{a}\rangle\,. \tag{147}$$

Consequently, we arrive at

$$|\Psi'_{\text{forest}}(p_{\text{trees}}, \mathbf{D}_{\text{train}})\rangle \equiv |\Psi'(\mathbf{D}_{\text{train}})\rangle \otimes \hat{U}_{\text{trees}}(p_{\text{trees}})\,|\mathbf{a}\rangle \tag{148}$$

for the total data encoding, where $\otimes$ denotes the tensor product.

In the second step, the structure of the forest is encoded using

$$\begin{aligned} |\Psi''_{\text{forest}}(F, p_{\text{trees}}, \mathbf{D}_{\text{train}})\rangle &= \hat{U}_{\text{forest}}(F)\,|\Psi'_{\text{forest}}(p_{\text{trees}}, \mathbf{D}_{\text{train}})\rangle \\ &= \sum_{\mathbf{x},\mathbf{y},\mathbf{a}} \sqrt{p_F(\mathbf{x}, \mathbf{y}, \mathbf{a})}\,|\mathbf{x}, \mathbf{y}, \mathbf{a}\rangle\,, \end{aligned} \tag{149}$$

which only acts on the feature and ancilla qubits, respectively, where

$$\hat{U}_{\text{forest}}(F) = \prod_{t=1}^{f} \prod_{i=0}^{d} \hat{U}^i_{\text{forest}}(\mathbf{c}^i(t), t) \tag{150}$$

iterates over all trees $1, \ldots, f$ and all layers $0, \ldots, d$. Here we also introduce

$$\hat{U}^i_{\text{forest}}(\mathbf{c}^i(t), t) \equiv \prod_{j=1}^{2^i} \hat{\Xi}^i_j(c^i_j(t), t) \tag{151}$$

in analogy to Eq. (45), where

$$\hat{\Xi}^i_j(c, t) \equiv \hat{\rho}(t)\, \hat{\mu}^i_j\, \hat{\xi}^i(c, t)\, \hat{\mu}^i_j\, \hat{\rho}(t) \tag{152}$$

with $c \in \{i+1, \ldots, k\}$ and we recall $\hat{\mu}^i_j$, Eq. (48). The products in Eqs. (150) and (151) are to be understood in the sense of Eq. (46). Furthermore, we use the abbreviations

$$\hat{\xi}^i(c, t) \equiv \begin{cases} \mathrm{C}_{f'+i}\mathrm{SWAP}(i+1, c, \{k+m+1, \ldots, k+m+f', 1, \ldots, i\}) & \text{for } i \geq 1 \\ \mathrm{C}_{f'}\mathrm{SWAP}(i+1, c, \{k+m+1, \ldots, k+m+f'\}) & \text{for } i = 0 \end{cases} \tag{153}$$

and

$$\hat{\rho}(t) \equiv \prod_{u=1}^{f'} \begin{cases} \mathrm{NOT}(u) & \text{for } \kappa(t, f', u) = 0 \\ \mathbb{1} & \text{otherwise} \end{cases} \tag{154}$$

for which we recall Eq. (51) and the standard gates NOT, $\mathrm{C}_v\mathrm{SWAP}$ (with $f' \leq v \leq f' + d$), and $\mathbb{1}$, respectively, from Sec. 4.1.2. Due to Eq. (153), the structural information of each tree is additionally conditioned by the ancilla qubits. An example of this structural encoding is shown in Fig. 20.

In the third and final measurement step, we simultaneously measure the first $d+1$ qubits (to obtain feature information), the qubits $k+1$ to $k+m$ (to obtain label information), and the qubits $k+m+1$ to $k+m+f'$ (to obtain ancilla information). Thus, we measure a bit string

$$\bar{x}_1 \cdots \bar{x}_{d+1} \bar{\mathbf{y}} \bar{\mathbf{a}} \sim p_F(\bar{x}_1, \ldots, \bar{x}_{d+1}, \bar{\mathbf{y}}, \bar{\mathbf{a}}) \tag{155}$$

similar to Eq. (55) with

$$\begin{aligned} & p_F(\bar{x}_1, \ldots, \bar{x}_{d+1}, \bar{\mathbf{y}}, \bar{\mathbf{a}}) \\ \equiv\ & p_F(x_1 = \bar{x}_1, \ldots, x_{d+1} = \bar{x}_{d+1}, \mathbf{y} = \bar{\mathbf{y}}, \mathbf{a} = \bar{\mathbf{a}}) \\ =\ & \mathrm{Tr}\{\langle \bar{x}_1, \ldots, \bar{x}_{d+1}, \bar{\mathbf{y}}, \bar{\mathbf{a}} | \Psi''_{\text{forest}}(F) \rangle \langle \Psi''_{\text{forest}}(F) | \bar{x}_1, \ldots, \bar{x}_{d+1}, \bar{\mathbf{y}}, \bar{\mathbf{a}} \rangle\}_{\bar{x}_{d+2}, \ldots, \bar{x}_k} \\ =\ & p_{\text{trees}}(\mathbf{a} = \bar{\mathbf{a}}) p_{T(\tau(\bar{\mathbf{a}}))}(\bar{x}_1, \ldots, \bar{x}_{d+1}, \bar{\mathbf{y}}), \end{aligned} \tag{156}$$

where we recall Eq. (56). Since the tree structure is additionally conditioned on the ancilla, a specific tree from the forest

$$\tau \equiv \tau(\bar{\mathbf{a}}) \equiv \sum_{j=1}^{f'} 2^{j-1} \bar{a}_{f'-j+1} + 1 \in \{1, \ldots, f\} \tag{157}$$

is evaluated depending on the measured ancilla values $\bar{\mathbf{a}}$.

To induce a forest, the individual trees are typically induced independently on a subsample of the data that is randomly and with replacement drawn from $\mathbf{D}_{\text{train}}$, a method also known as *bootstrap aggregating* or *bagging*. In addition, random subsets of features can optionally be selected for each tree, a method known as feature bagging. The method described in Sec. 4.2 can therefore be applied for the induction of the Q-forest in the sense of a Q-tree ensemble, where each Q-tree is trained using its own data set. To determine the distribution of trees p_{trees}, the relative performance of the previously induced trees can be determined (e. g., using the prediction accuracy

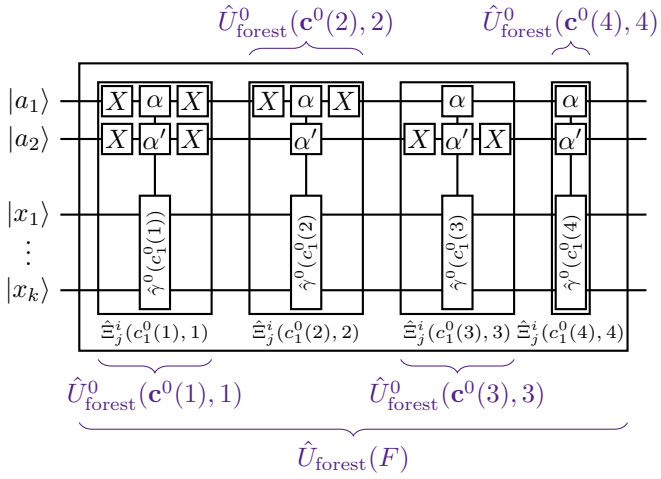


Figure 20: Example of the structural encoding of a decision forest consisting of $f = 4$ trees of maximum depth $d = 0$, which requires $f' = 2$ ancilla qubits, Eq. (146). The structure is encoded by the unitary operator $\hat{U}_{\text{forest}}(F)$, Eq. (150), as defined in Eq. (149). Specifically, $\hat{U}^i_{\text{forest}}(\mathbf{c}^i(t), t)$ from Eq. (151) encodes the structure of the tree $t \in \{1, \ldots, f\}$ in depth $i \in \{0, \ldots, d\}$ defined by the corresponding compressed decision configuration $\mathbf{c}^i(t)$, Eq. (13). It consists of a composition of unitary operators $\hat{\Xi}^i_j(c, t)$, Eq. (152), which in turn consist of NOT gates and C_2SWAP gates, respectively. These C_2SWAP gates are effectively generalizations of $\hat{\gamma}^i(c)$, Eq. (49), which are additionally conditioned on the ancilla qubits. We denote NOT gates with X and mark the additional control qubits of C_2SWAP gates with α and α', respectively, in analogy to the notation from Fig. 4 such that $\hat{\xi}^0(c, t) = C_2\text{SWAP}(1, c, \{\alpha, \alpha'\})$, Eq. (153). We also explicitly show consecutive NOT gates for the sake of clarity.

of test data). Appropriately normalized, it can be directly used as a probability distribution such that trees with a high performance correlate with a high probability [84].

A forest prediction can be performed by taking the weighted average of all tree predictions following the scheme in Sec. 4.3. For a query request $\mathbf{x}^q \in \mathbb{B}^k$ and N measurements, the measurement counts of a bit string of ancilla qubits $n(\mathbf{a})$ allow to estimate the tree distribution according to

$$p_{\text{trees}}(\mathbf{a}) \approx \hat{p}_{\text{trees}}(\mathbf{a}) \equiv \frac{n(\mathbf{a})}{N} \tag{158}$$

and can also be used together with the other parts of the measured bit strings to estimate the corresponding label probability distribution $\hat{p}_{T(\tau(\mathbf{a}))}(y_i \,|\, \mathcal{C}_\nu)$, Eq. (32), for all $\nu \in \{1, \ldots, 2^d\}$ and all trees $\tau \in \{1, \ldots, f\}$ according to Eq. (61). Here we recall the set of conditions $\mathcal{C}_\nu$, Eq. (28). Thus, we can obtain the Q-forest prediction

$$\hat{p}_{\text{cl}}(y_i \,|\, \mathbf{x}^q) \equiv \sum_{\mathbf{a}} \hat{p}_{\text{forest}}(\bar{\mathbf{a}}) \hat{p}_{T(\tau(\mathbf{a}))}(y_i \,|\, \mathcal{C}_{q_d(\mathbf{x}^q)}) \tag{159}$$

as a generalization of Eq. (63), where we recall $\mathcal{C}_{q_d(\mathbf{x}^q)}$, Eq. (38).

References

[1] S.B. Kotsiantis. "Decision trees: a recent overview". In: *Artif Intell Rev* 39 (2013), pp. 261–283. DOI: 10.1007/s10462-011-9272-4.

[2] O.Z. Maimon and L. Rokach. *Data Mining With Decision Trees: Theory And Applications*. 2nd. Series In Machine Perception And Artificial Intelligence. World Scientific Publishing Company, 2014. ISBN: 9789814590099.

[3] L. Breiman. *Classification and Regression Trees*. CRC Press, 2017. ISBN: 9781351460491.

[4] Laurent Hyafil and Ronald L. Rivest. “Constructing optimal binary decision trees is NP-complete”. In: *Information Processing Letters* 5.1 (1976), pp. 15–17. ISSN: 0020-0190. DOI: 10.1016/0020-0190(76)90095-8.

[5] Dimitris Bertsimas and Jack Dunn. “Optimal classification trees”. In: *Machine Learning* 106.7 (June 2017), pp. 1039–1082. ISSN: 1573-0565. DOI: 10.1007/s10994-017-5633-9.

[6] Arman Zharmagambetov et al. *An Experimental Comparison of Old and New Decision Tree Algorithms*. 2020. arXiv: 1911.03054 [cs.LG].

[7] J.R. Quinlan. “Induction of decision trees”. In: *Machine Learning* 1 (1986), pp. 81–706. DOI: 10.1007/BF00116251.

[8] Jacob Biamonte et al. “Quantum machine learning”. In: *Nature* 549.7671 (Sept. 2017), pp. 195–202. ISSN: 1476-4687. DOI: 10.1038/nature23474.

[9] Edward Farhi and Sam Gutmann. “Quantum computation and decision trees”. In: *Phys. Rev. A* 58 (2 Aug. 1998), pp. 915–928. DOI: 10.1103/PhysRevA.58.915.

[10] Songfeng Lu and Samuel L. Braunstein. “Quantum decision tree classifier”. In: *Quantum Information Processing* 13.3 (Mar. 2014), pp. 757–770. ISSN: 1573-1332. DOI: 10.1007/s11128-013-0687-5.

[11] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. “An introduction to quantum machine learning”. In: *Contemporary Physics* 56.2 (2015), pp. 172–185. DOI: 10.1080/00107514.2014.964942.

[12] Kamil Khadiev, Ilnaz Mannapov, and Liliya Safina. *The Quantum Version Of Classification Decision Tree Constructing Algorithm C5.0*. 2019. arXiv: 1907.06840 [cs.LG].

[13] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. “Quantum Random Access Memory”. In: *Physical Review Letters* 100.16 (Apr. 2008). ISSN: 1079-7114. DOI: 10.1103/physrevlett.100.160501.

[14] Iordanis Kerenidis and Anupam Prakash. *Quantum Recommendation Systems*. 2016. arXiv: 1603.08675 [quant-ph].

[15] Howard Barnum, M Saks, and M Szegedy. *Quantum Decision Trees and Semidefinite Programming*. Jan. 2001. URL: https://www.osti.gov/biblio/975874.

[16] Harry Buhrman and Ronald de Wolf. “Complexity measures and decision tree complexity: a survey”. In: *Theoretical Computer Science* 288.1 (2002). Complexity and Logic, pp. 21–43. ISSN: 0304-3975. DOI: 10.1016/S0304-3975(01)00144-X.

[17] Yaoyun Shi. “Entropy lower bounds for quantum decision tree complexity”. In: *Information Processing Letters* 81.1 (2002), pp. 23–27. ISSN: 0020-0190. DOI: 10.1016/S0020-0190(01)00191-0.

[18] Salman Beigi and Leila Taghavi. “Quantum Speedup Based on Classical Decision Trees”. In: *Quantum* 4 (Mar. 2020), p. 241. ISSN: 2521-327X. DOI: 10.22331/q-2020-03-02-241.

[19] Cristian S. Calude et al. “Experimental evidence of quantum randomness incomputability”. In: *Physical Review A* 82.2 (Aug. 2010). ISSN: 1094-1622. DOI: 10.1103/physreva.82.022102.

[20] Johannes Kofler and Anton Zeilinger. “Quantum Information and Randomness”. In: *European Review* 18.4 (Oct. 2010), pp. 469–480. ISSN: 1474-0575. DOI: 10.1017/s1062798710000268.

[21] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. “A review on genetic algorithm: past, present, and future”. In: *Multimedia Tools and Applications* 80.5 (Feb. 2021), pp. 8091–8126. ISSN: 1573-7721. DOI: 10.1007/s11042-020-10139-6.

[22] Guang Hao Low, Theodore J. Yoder, and Isaac L. Chuang. “Quantum inference on Bayesian networks”. In: *Phys. Rev. A* 89 (6 June 2014), p. 062315. DOI: 10.1103/PhysRevA.89.062315.

[23] Sima E. Borujeni et al. “Experimental evaluation of quantum Bayesian networks on IBM QX hardware”. In: *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. 2020, pp. 372–378. DOI: 10.1109/QCE49297.2020.00053.

[24] Michael de Oliveira and Luis Soares Barbosa. “Quantum Bayesian Decision-Making”. In: *Foundations of Science* (Mar. 2021). ISSN: 1572-8471. DOI: 10.1007/s10699-021-09781-6.

[25] S. Akers. "Binary Decision Diagrams". In: *IEEE Transactions on Computers* 27.06 (June 1978), pp. 509–516. ISSN: 1557-9956. DOI: 10.1109/TC.1978.1675141.

[26] Foster Provost and Pedro Domingos. "Tree Induction for Probability-Based Ranking". In: *Machine Learning* 52.3 (Sept. 2003), pp. 199–215. ISSN: 1573-0565. DOI: 10.1023/A:1024099825458.

[27] Melanie Mitchell. *An Introduction to Genetic Algorithms*. 5th ed. A Bradford book. MIT Press, Cambridge, 1999. ISBN: 0262133164.

[28] Alvaro H. C. Correia, Robert Peharz, and Cassio de Campos. *Joints in Random Forests*. 2020. arXiv: 2006.14937 [cs.LG].

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

[30] Lov Grover and Terry Rudolph. *Creating superpositions that correspond to efficiently integrable probability distributions*. 2002. arXiv: 0208112 [quant-ph].

[31] Maris Ozols, Martin Roetteler, and Jérémie Roland. "Quantum rejection sampling". In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference on - ITCS'12* (2012). DOI: 10.1145/2090236.2090261.

[32] Maria Schuld and Francesco Petruccione. *Supervised Learning with Quantum Computers*. Quantum Science and Technology. Springer International Publishing, 2018.

[33] Mikko Möttönen et al. "Transformation of Quantum States Using Uniformly Controlled Rotations". In: *Quantum Info. Comput.* 5.6 (Sept. 2005), pp. 467–473. ISSN: 1533-7146.

[34] V.V. Shende, S.S. Bullock, and I.L. Markov. "Synthesis of quantum-logic circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.6 (June 2006), pp. 1000–1010. ISSN: 1937-4151. DOI: 10.1109/tcad.2005.855930.

[35] Martin Plesch and Časlav Brukner. "Quantum-state preparation with universal gate decompositions". In: *Physical Review A* 83.3 (Mar. 2011). ISSN: 1094-1622. DOI: 10.1103/physreva.83.032302.

[36] Yuval R. Sanders et al. "Black-Box Quantum State Preparation without Arithmetic". In: *Physical Review Letters* 122.2 (Jan. 2019). ISSN: 1079-7114. DOI: 10.1103/physrevlett.122.020502.

[37] Johannes Bausch. *Fast Black-Box Quantum State Preparation*. 2021. arXiv: 2009.10709 [quant-ph].

[38] Pierre-Luc Dallaire-Demers and Nathan Killoran. "Quantum generative adversarial networks". In: *Physical Review A* 98.1 (July 2018). ISSN: 2469-9934. DOI: 10.1103/physreva.98.012324.

[39] Marcello Benedetti et al. "Parameterized quantum circuits as machine learning models". In: *Quantum Science and Technology* 4.4 (Nov. 2019), p. 043001. ISSN: 2058-9565. DOI: 10.1088/2058-9565/ab4eb5.

[40] Ling Hu et al. "Quantum generative adversarial learning in a superconducting quantum circuit". In: *Science Advances* 5.1 (Jan. 2019). ISSN: 2375-2548. DOI: 10.1126/sciadv.aav2761.

[41] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. "Quantum Generative Adversarial Networks for learning and loading random distributions". In: *npj Quantum Information* 5.1 (Nov. 2019), p. 103. ISSN: 2056-6387. DOI: 10.1038/s41534-019-0223-2.

[42] Adriano Barenco et al. "Elementary gates for quantum computation". In: *Physical Review A* 52.5 (Nov. 1995), pp. 3457–3467. ISSN: 1094-1622. DOI: 10.1103/physreva.52.3457.

[43] Israel F. Araujo et al. "A divide-and-conquer algorithm for quantum state preparation". In: *Scientific Reports* 11.1 (Mar. 2021), p. 6329. ISSN: 2045-2322. DOI: 10.1038/s41598-021-85474-1.

[44] Xiaoming Sun et al. *Asymptotically Optimal Circuit Depth for Quantum State Preparation and General Unitary Synthesis*. 2021. arXiv: 2108.06150 [quant-ph].

[45] Xiao-Ming Zhang, Man-Hong Yung, and Xiao Yuan. *Low-depth Quantum State Preparation*. 2021. arXiv: 2102.07533 [quant-ph].

[46] Ji Liu, Luciano Bello, and Huiyang Zhou. *Relaxed Peephole Optimization: A Novel Compiler Optimization for Quantum Circuits*. 2020. arXiv: 2012.07711 [quant-ph].

[47] Laxmidhar Biswal et al. “Techniques for fault-tolerant decomposition of a multicontrolled Toffoli gate”. In: *Physical Review A* 100.6 (Dec. 2019). ISSN: 2469-9934. DOI: 10.1103/physreva.100.062326.

[48] Mehdi Saeedi and Massoud Pedram. “Linear-depth quantum circuits forn-qubit Toffoli gates with no ancilla”. In: *Physical Review A* 87.6 (June 2013). ISSN: 1094-1622. DOI: 10.1103/physreva.87.062318.

[49] Manabendra Nath Bera et al. “Randomness in quantum mechanics: philosophy, physics and technology”. In: *Reports on Progress in Physics* 80.12 (Nov. 2017), p. 124001. ISSN: 1361-6633. DOI: 10.1088/1361-6633/aa8731.

[50] M. Cerezo et al. *Variational Quantum Algorithms*. 2020. arXiv: 2012.09265 [quant-ph].

[51] W. Zhai, P. Kelly, and W.-B. Gong. “Genetic algorithms with noisy fitness”. In: *Mathematical and Computer Modelling* 23.11 (1996), pp. 131–142. ISSN: 0895-7177. DOI: 10.1016/0895-7177(96)00068-4.

[52] Donald A. Sofge. “Toward a Framework for Quantum Evolutionary Computation”. In: *2006 IEEE Conference on Cybernetics and Intelligent Systems*. 2006, pp. 1–6. DOI: 10.1109/ICCIS.2006.252360.

[53] Gexiang Zhang. “Quantum-inspired evolutionary algorithms: a survey and empirical study”. In: *Journal of Heuristics* 17.3 (June 2011), pp. 303–351. ISSN: 1572-9397. DOI: 10.1007/s10732-010-9136-0.

[54] Rafael Lahoz-Beltra. “Quantum Genetic Algorithms for Computer Scientists”. In: *Computers* 5.4 (2016). ISSN: 2073-431X. DOI: 10.3390/computers5040024.

[55] Harper R. Grimsley et al. “An adaptive variational algorithm for exact molecular simulations on a quantum computer”. In: *Nature Communications* 10.1 (June 2019), p. 3007. ISSN: 2041-1723. DOI: 10.1038/s41467-019-10988-2.

[56] Arthur G. Rattew et al. *A Domain-agnostic, Noise-resistant, Hardware-efficient Evolutionary Variational Quantum Eigensolver*. 2020. arXiv: 1910.09694 [quant-ph].

[57] L. Franken et al. *Gradient-free quantum optimization on NISQ devices*. 2021. arXiv: 2012.13453 [quant-ph].

[58] Andrew Arrasmith et al. *Effect of barren plateaus on gradient-free optimization*. 2020. arXiv: 2011.12245 [quant-ph].

[59] David W. Aha. *Tic-Tac-Toe Endgame Data Set*. accessed June 2021. 1991. URL: https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame.

[60] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[61] Héctor Abraham et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: 10.5281/zenodo.2562110.

[62] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[63] John Preskill. “Quantum Computing in the NISQ era and beyond”. In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79.

[64] IBM. *IBM Quantum*. 2021. URL: https://quantum-computing.ibm.com.

[65] Ryan LaRose. “Overview and Comparison of Gate Level Quantum Software Platforms”. In: *Quantum* 3 (Mar. 2019), p. 130. ISSN: 2521-327X. DOI: 10.22331/q-2019-03-25-130.

[66] Andrew W. Cross et al. “Validating quantum computers using randomized model circuits”. In: *Physical Review A* 100.3 (Sept. 2019). ISSN: 2469-9934. DOI: 10.1103/physreva.100.032328.

[67] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324.

[68] Chin-Yao Chang, Eric Jones, and Peter Graf. *On Quantum Computing for Mixed-Integer Programming*. 2020. arXiv: 2010.07852 [math.OC].

[69] Jin-Kao Kochenberger Garyand Hao et al. “The unconstrained binary quadratic programming problem: a survey”. In: *Journal of Combinatorial Optimization* 28.1 (July 2014), pp. 58–81. ISSN: 1573-2886. DOI: 10.1007/s10878-014-9734-0.

[70] Ehsan Zahedinejad and Arman Zaribafiyan. *Combinatorial Optimization on Gate Model Quantum Computers: A Survey*. 2017. arXiv: 1708.05294 [quant-ph].

[71] C. E. Shannon. “A mathematical theory of communication”. In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[72] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley, 2012. ISBN: 9781118585771.

[73] Nisha Saini. “Review of Selection Methods in Genetic Algorithms”. In: *International Journal of Engineering and Computer Science* 6.12 (Dec. 2017), pp. 22261–22263. URL: http://www.ijecs.in/index.php/ijecs/article/view/2562.

[74] G. Cowan. “Statistics”. In: *Review of Particle Physics*. Ed. by P A Zyla et al. Progress of Theoretical and Experimental Physics 8. Oxford University Press, 2020, pp. 626–641. DOI: 10.1093/ptep/ptaa104. eprint: https://academic.oup.com/ptep/article-pdf/2020/8/083C01/34673740/rpp2020-vol2-2015-2092_18.pdf.

[75] Jerzy Neyman. *A selection of early statistical papers*. The Selected Papers of Jerzy Neyman and E. S. Pearson. University of California Press, 1967. ISBN: 9780520009929.

[76] Niek J. Bouman and Serge Fehr. *Sampling in a Quantum Population, and Applications*. 2012. arXiv: 0907.4246 [quant-ph].

[77] Lawrence D. Brown, T. Tony Cai, and Anirban DasGupta. “Interval Estimation for a Binomial Proportion”. In: *Statistical Science* 16.2 (2001), pp. 101–133. DOI: 10.1214/ss/1009213286.

[78] Abdullah Ash Saki, Mahabubul Alam, and Swaroop Ghosh. *Study of Decoherence in Quantum Computers: A Circuit-Design Perspective*. 2019. arXiv: 1904.04323 [cs.ET].

[79] G. Lindblad. “On the generators of quantum dynamical semigroups”. In: *Communications in Mathematical Physics* 48.2 (June 1976), pp. 119–130. ISSN: 1432-0916. DOI: 10.1007/BF01608499.

[80] B. M. Villegas-Martínez, F. Soto-Eguibar, and H. M. Moya-Cessa. “Application of Perturbation Theory to a Master Equation”. In: *Advances in Mathematical Physics* 2016 (June 2016), p. 9265039. ISSN: 1687-9120. DOI: 10.1155/2016/9265039.

[81] Kamil Khadiev and Liliya Safina. “The Quantum Version of Random Forest Model for Binary Classification Problem”. In: *YRID-2020: International Workshop on Data Mining and Knowledge Engineering*. Oct. 2020, pp. 30–35.

[82] Maria Schuld and Francesco Petruccione. “Quantum ensembles of quantum classifiers”. In: *Scientific Reports* 8.1 (Feb. 2018), p. 2772. ISSN: 2045-2322. DOI: 10.1038/s41598-018-20403-3.

[83] Zeke Xie and Issei Sato. *A Quantum-Inspired Ensemble Method and Quantum-Inspired Forest Regressors*. 2017. arXiv: 1711.08117 [cs.LG].

[84] Mohsen Shahhosseini and Guiping Hu. “Improved Weighted Random Forest for Classification Problems”. In: *Progress in Intelligent Decision Science* (2021), pp. 42–56. ISSN: 2194-5365. DOI: 10.1007/978-3-030-66501-2˙4.