Universiteit
Leiden
The Netherlands

Separating quantum and classical computing: rigorous proof and practical application

Marshall, S.C.

Citation
Marshall, S. C. (2025, May 27). *Separating quantum and classical computing: rigorous proof and practical application*. Retrieved from https://hdl.handle.net/1887/4247215

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/4247215

Note: To cite this publication please use the final published version (if applicable).

CHAPTER 1

Introduction

In recent decades, quantum algorithms have been discovered that are exponentially faster than our best-known classical algorithms. However, the quantifier *best-known* is needed, as we do not know if quantum computers are fundamentally faster on these tasks or if we just haven't discovered equivalently fast classical algorithms *yet*. This thesis addresses this disconnect, firstly by expanding our theoretical understanding of the consequences if quantum computers and classical computers are equivalently powerful, and then by analysing multiple algorithms which may allow us to make better use of quantum computers deployable today or in the near future.

Chapter 1 provides a set of definitions of the mathematical tools and definitions (such as quantum computing, various complexity classes, and machine learning) we will use in later chapters. For concepts used in only one chapter, we leave definitions to those chapters.

Chapter 2 provides the strongest evidence that quantum computers can implement functions or sampling from distributions that classical computers cannot.

Motivated by understanding the separation between quantum and classical methods in machine learning we develop the concept of bounded advice in Chapter 3, i.e. enhancing some limited Turing machines with advice given by a stronger machine. This concept allows us to understand if quantum computers used to prepare advice (such as during training) can

provide an edge over classical machine learning even when the ultimate model is entirely classical. Bounded advice proves to have utility beyond this, being useful for understanding situations in cryptography and for general inquiry into complexity theory.

The next chapters develop methods to make better use of near-term machines. For the following two chapters we focus on a technique called circuit cutting, which reduces the number of qubits needed for a quantum computation at the cost of an exponentially increasing number of evaluations of quantum circuits.

Chapter 4 proposes a new quantum machine learning algorithm. The algorithm combines many smaller quantum circuits, just as circuit cutting does, but allows for the setting of the number of circuits as a hyperparameter. Lower circuit counts yield an algorithm that could be practically used, higher circuit counts approximate the full circuit cutting scheme. We analyse the learning performance of this model and experimentally test it, this allows us to compare quantum and classical computing more large-scale quantum machines are available.

Chapter 5 continues the above research line by proving that while it may be possible to reduce the number of smaller circuits produced by cutting a larger circuit for certain circuits, this is not the case in general. We prove that if any *cut-local*[1] cutting scheme can remove even a single qubit from a circuit without running an exponential (in number of gates such a cut would entail) number of smaller circuits, then all decision problems solvable by a quantum computer can also be solved by a classical computer.

In the Chapter 6, we develop a quantum machine learning algorithm which can be deployed on a classical computer while still reaping the benefits of a quantum computer to train. This is formally linked to quantum-generated classical advice, as treated in Chapter 2.

[1] this will be defined in Chapter 5

1.1 Quantum Computing

To define quantum computation we first state the postulates of quantum mechanics. Quantum mechanics is commonly defined by 4 postulates. We provide these postulates using bra-ket notation for vectors [1].

1. The state space of any physical system is a Hilbert space. The system is entirely described by a state vector in this Hilbert space.

2. The evolution of a closed quantum system over a fixed amount of time can be described by a unitary transform acting on that system's Hilbert space.

3. The measurement of a system is defined by a set of linear measurement operators $\{M_m\}$. The probability of measuring outcome m for state $|\psi\rangle$ is $p(m) = \langle\psi| M_m^\dagger M |\psi\rangle$. The state after measurement outcome m is $M_m\psi/\sqrt{p(m)}$.

4. The composition of multiple physical systems is given by the tensor product of their associated Hilbert spaces. The composed state of these physical systems is the tensor product of the individual states.

These postulates define the rules of quantum mechanics. This thesis concerns quantum computing, which is an attempt to improve computational efficiency by using the increased capabilities provided by these rules, as compared to those provided by the rules of classical physics.

In quantum computing, we are often only concerned with the quantum bit, as a minimum-sized unit of information which is always a linear combination of 0 and 1. Here we use bra-ket notation and $\alpha, \beta \in \mathbb{C}$ for $|\alpha|^2 + |\beta|^2 = 1$

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

Larger systems (such as a computer) can be described as the combination of individual qubits following postulate 4.

Before we define quantum computing, we must define gate sets and circuits. A gate set is a set of unitaries we can apply on our quantum computer, due to the infinite nature of the space of unitaries there is an infinite set of possible gate sets. Any particular quantum device will have some gates that are more natural than others. Fortunately, some gate sets are universal, that is they can approximate any unitary operation on a quantum system to arbitrary precision. For the purpose of theory, it is therefore sufficient to define a single universal gate set and rely on the

ability of other universal gate sets (that may be native to any given device) to approximate our circuit with high accuracy.

The following gates are defined using the vector notation of quantum states ($|0\rangle = (1,0)^T$, $|1\rangle = (0,1)^T$).

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

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

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

It is often easiest to describe a quantum circuit pictographically, for single-qubit gates this is normally given by a white box with the name of the gate inside of it. For some 2 qubit gates, we have less direct representations, the CNOT is represented by the following symbol.

(1.1)

The 3 above gates form a universal gate set. Depending on the circumstance it can be easier to write operations with other gates instead, such as with the following "rotation" gates. Of course, these gates could be equally approximated by the above gates, but this would require much more complicated notation for some situations.

$$R_x(\theta) := \quad -\boxed{R_x(\theta)}- \quad = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \tag{1.2}$$

$$R_y(\theta) := \quad -\boxed{R_y(\theta)}- \quad = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \tag{1.3}$$

$$R_z(\theta) := \quad -\boxed{R_z(\theta)}- \quad = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix} \tag{1.4}$$

1.1.1 Limited modern machines

The previous section described an idealised quantum computation. In the real world, we are not able to perfectly execute quantum circuits; noise is naturally introduced that destroys quantum superposition [1]. Fortunately, if noise is suppressed below a threshold error correction is possible, where many qubits are used to simulate one nearly-perfect or *logical* qubit. Unfortunately, this process of error correction often requires a large number of physical qubits to produce one logical qubit, and a large overhead on the number of physical gates needed to implement one logical gate. Thus it is often considered how we could perform interesting computations while avoiding costly error correction with our modern Noisy Intermediate Scale Quantum computers (NISQ) [2].

There are many interesting approaches to dealing with noise in quantum computing without error correction, but this work instead tackles the other component of NISQ: intermediate scale. In the following subsection, we outline one such method, circuit cutting.

1.1.2 Circuit Cutting

Circuit cutting [3–11], sometimes referred to as circuit partitioning or circuit knitting, is a set of techniques to take one large quantum circuit and to estimate its output by measuring many smaller quantum circuits. Existing schemes either make multiple calls to the device [3–6] or link multiple devices with classical communication [7, 9] to simulate the larger circuit.

Each of the existing circuit-cutting schemes works slightly differently, but they all take a local element of the large circuit and represent it as the sum of terms, which can then be separated into smaller circuits. We will give a generalization of circuit cutting by cutting the unitaries. This generalization involves decomposing a unitary into the sum of several smaller unitaries. Here, decomposing means expressing a given n-qubit unitary as a sum of tensor products of two fewer-qubit unitaries:

$$U = \sum_{i}^{L} \alpha_i U_i' \otimes U_i'', \tag{1.5}$$

As every 2 qubit unitary matrix is equal to the sum of (at most) 4 single qubit tensor product matrices this scheme can break any connecting elements across a large circuit and allow the two chunks to be evaluated independently. This is shown in figure 1.1.

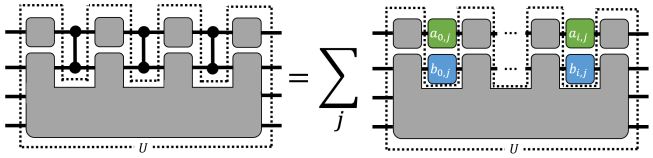

Figure 1.1: Figure depicts circuit cutting applied to a quantum circuit. By separating the three controlled-Z gates into the sum of a polynomial number of single qubit unitary we produce a bottom circuit with $n-1$ qubits and a top circuit consisting of one qubit, which can be run independently, lowering the required number of qubits to evaluate the original circuit.

This circuit-cutting generalisation most closely resembles the original proposal [12]. Other circuit cutting schemes do not necessarily fit this unitary cutting generalisation but cut some other element of the circuit. While these schemes do not fit within the generalisation presented we will see that the theorems we develop for unitary cutting carry over to these cases as they are not conceptually different.

1.2 Machine Learning

Machine learning is a set of statistical methods that take data from some problem and, by adjusting internal weights, are able to generalise to solve that problem at large. There are many fields nested inside machine learning, such as supervised, unsupervised and reinforcement learning. For the sake of this thesis, we will only be interested in supervised learning, i.e. when the problem is to label data in a similar manner to labelled examples.

To define supervised learning we will break it down into a number of separate parts: the task, the model and the training procedure.

Let us begin by formalising the task. We define a supervised learning task on a domain (inputs) $\mathcal{X}$ and co-domain (outputs) $\mathcal{Y}$ with a probability distribution (the probability of each input and output pair) over $\mathcal{X} \times \mathcal{Y}$, P, and loss function, $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.

To formalise the model, we first define a hypothesis, $h : \mathcal{X} \to \mathcal{Y}$. The hypothesis defines a map from input data points to output labels, this definition does not allow for probabilistic labelling but that can be included in more complicated definitions. Writing out a truth table for h would be cumbersome. Instead, we often define some parameterised model, such as a neural network, that has some set of parameters we can tune, modifying

these parameters gives us access to different hypotheses.

Formalising the training procedure is more complicated as there are many potential training methods in SL (supervised learning). Broadly the training objective is to find parameters that allow the model to perform well on the task. Performing well is often defined by minimising the expected value of the loss function on the output of the model and the correct label. If this distance is small, the model is mostly correct most of the time. So the training procedure's task is to minimise the expected amount of loss[2]. Define the risk for a hypothesis h on a space of continuous inputs/outputs as the expected loss:

$$R(h) = \int_{\mathcal{X}\times\mathcal{Y}} \ell(h(x), y) dP(x, y). \tag{1.6}$$

The ultimate objective is to minimise this loss.

In practical settings, we normally lack access to the underlying probability distribution, P, so the *true* risk cannot be evaluated. Instead, we have a data set, this data is drawn from P but is only a sample, $S = \{(x_i, y_i) \sim P \,|\, i \in [m]\}$. This dataset makes it possible to approximate the risk, we call this the empirical risk of h with respect to S:

$$\hat{R}_S(h) = \frac{1}{|S|} \sum_{(x_i,y_i)\in S} \ell(h(x_i), y_i). \tag{1.7}$$

Optimising our hypothesis on the training data optimises the empirical risk, which is generally a good proxy for the true risk, but it is possible that these two diverge, this is sometimes called 'generalisation error' and can be characterised by 'over/underfitting'. More about this divergence will be said in Chapter 4 where showing the model defined in that section has a small generalisation loss is of key interest.

1.3 Complexity Theory

Large parts of this thesis are written in terms of complexity classes, loosely defined as sets of problems which can be solved by different levels of computational ability. We assume the reader is familiar with the standard definitions of Turing machines and focus here on establishing the notation and variant models used throughout this work.

[2]It is possible other training objectives could be given.

1.3.1 Computational Models

The *deterministic Turing machine* (DTM) [13] serves as the standard theoretical formalisation of computation. This thesis assumes familiarity with the DTM, but we will also use some standard variants:

A *non-deterministic Turing machine* (NTM) [13] extends the deterministic model by allowing any given state of the Turing machine to lead to a number of possible next states and actions. The NTM is said to accept an input if there exists at least one sequence of non-deterministic choices leading to an accepting state.

A *probabilistic Turing machine* (PTM) [13] builds upon the non-deterministic model by assigning probabilities to each possible transition. While an NTM simply explores all possible computational paths, a PTM randomly selects transitions according to their associated probabilities, providing a framework for randomised algorithms and error bounds in computation. The acceptance criteria for a PTM are given by the particular complexity class it is used in.

A *quantum Turing machine* (QTM) [1] incorporates quantum mechanics into the computational model, allowing configurations to exist in superpositions and employing quantum operations (unitary transformations and measurements) for transitions.

An *oracle Turing machine* [13] augments the standard model with an external 'oracle' that can decide membership in some language L in a single step. These machines are central to defining relativized complexity classes like NP^{A}, where A represents the oracle, and help establish relationships between different complexity classes.

1.3.2 Formal Languages

A *formal language* is a set of strings over a finite alphabet Σ. Formally, a language $L \subseteq \Sigma^*$ is a set of strings that can be recognized by a computational model, such as a Turing machine. A decision problem can be framed as a language by considering the set of all "yes" instances as the strings that belong to the language. For example, the decision problem "Is a number prime?" can be represented as a language where the strings are the binary representations of prime numbers.

1.3.3 Complexity classes

Using the tools of the previous subsections we can now define a 'zoo' of complexity classes.

- P is the class of decision problems, $\{L \subset \{0,1\}^*\}$, that a deterministic Turing machine can determine membership of in time polynomial in the length of x.
- NP is the class of decision problems solvable by a non-deterministic polynomial-time Turing machine that accepts if at least one path accepts.
- CoNP is the complement of NP.
- We define levels of the polynomial hierarch with the following notation $\Sigma_0^P = \Pi_0^P = \Delta_0^P := P$. $\Delta_{i+1}^P := P^{\Sigma_i^P}$. $\Sigma_{i+1}^P := NP^{\Sigma_i^P}$. $\Pi_{i+1}^P := CoNP^{\Sigma_i^P}$[3].
- PH: The polynomial hierarchy is defined as $PH = \bigcup_i \Sigma_i^P$.
- BPP is the class of decision problems solvable by a probabilistic Turing machine with at most a 1/3 probability of error both for $x \in L$ and $x \notin L$.
- ZPP is the zero error alternative to BPP. A language is in ZPP if there exists a probabilistic Turing machine which either refuses to answer (which it does with less than 1/3 probability) or correctly determines if $x \in L$.
- EXP is the class of decision problems solvable by a Turing machine running in $O(2^{p(n)})$ for any polynomial $p(n)$.
- PP is the class of decision problems solvable by a probabilistic Turing machine with a probability of error less than 1/2.
- AM is the class of decision problems for which $x \in L$ can be verified by a probabilistic check: L is in AM if there exists a probabilistic polynomial time verifier, V, such that if $x \in L$ then, with at least 2/3 probability there exists a proof, y, such that V accepts. If $x \notin L$, with at least 2/3 probability, regardless of y, V must reject.
- BQP is the class of decision problems solvable by a quantum Turing machine with at most a 1/3 probability of error.
- QMA is the class of decision problems such that there exists a 'verifier' quantum Turing machine which takes the input and a quantum state 'proof'. For all yes inputs there exists a quantum state that leads the verifier to accept with probability at least 2/3. For all no inputs, all states lead the verifier to reject with probability at least 2/3.

[3]The notation $\_^P$ signals the polynomial hierarchy, not an oracle to P.

- P/poly is the class of decision problems solvable by a family of polynomial-size circuits. Equivalently P/poly is the class of decision problems solvable by a polynomial time deterministic Turing machine with polynomial length *advice*. Advice is an additional input string which may depend on the size of the input, but not on the input itself. Chapter 3 will provide more information on advice and it's associated concepts.

In complexity theory, a functional problem is defined by a relation, R. An algorithm solves this problem if, for all inputs, x, the algorithm outputs y such that $(x, y) \in R$ or halts if no such y exists. As with decision classes, there are a number of functional complexity classes we will need to define.

- #P is the class of functional problems that consist of computing the number of accepting paths of a non-deterministic Turing machine.
- FBPP is the class of functional problems that can be solved by a polynomial time probabilistic Turing machine which achieves any failure probability $\epsilon > 0$ given input $x, 0^{\lceil 1/\epsilon \rceil}$.
- FBQP is the class of functional problems that can be solved by a polynomial-time quantum Turing machine which achieves any failure probability $\epsilon > 0$ given input $x, 0^{\lceil 1/\epsilon \rceil}$[4].

While it is well known that functional and sampling problems are closely related [14], we will still make reference to the following sampling problems. A sampling problem is defined by a function that maps each input, x, to a probability distribution D_x. An algorithm solves the problem if, on input x, it outputs y sampled from close to D_x (where close is defined for each individual class).

- SampP is the class of sampling problems, $S = (\mathcal{D})_{x \in \{0,1\}^*}$ for which there exists a polynomial-time probabilistic Turing machine which given input $x, 0^{\lceil 1/\epsilon \rceil}$ outputs a distribution ϵ close to the problem's distribution in total-variation distance.
- SampBQP is the class of sampling problems for which there exists a polynomial-time quantum Turing machine which given input $x, 0^{\lceil 1/\epsilon \rceil}$ outputs a distribution ϵ close to the problem's distribution in total-variation distance.

[4] the notation 0^n refers to a string of n 0s

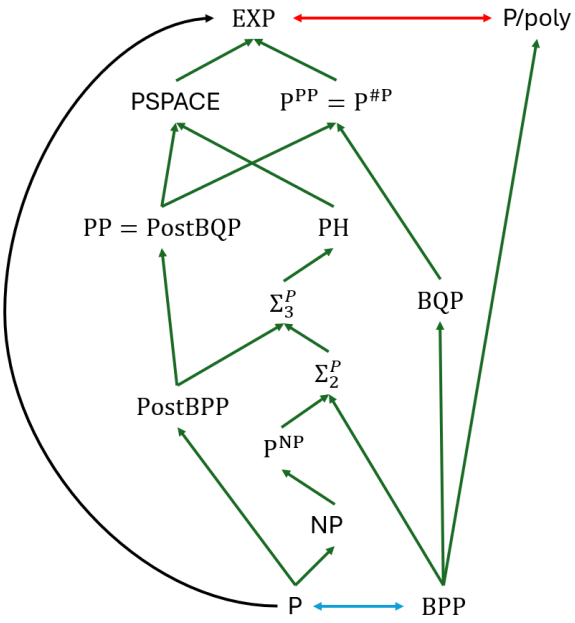

Figure 1.2: A diagrammatic representation of various complexity classes which contain each other. $\rightarrow$ represents a known inclusion that we believe to be strict. $\leftrightarrow$ represents classes we believe are unequal, while $\leftrightarrow$ represents two classes we believe to be equal. $\rightarrow$ is a known strict inclusion.

One particularly relevant problem in SampBQP is BosonSampling. This problem can be defined with reference to a *Bosonic computer* [15], but it is perhaps clearer to use the equivalent mathematical definition. For a matrix $A \in \mathbb{C}^{m \times n}$ and string of numbers $x \in \mathbb{N}^m$ such that $\sum x_i = n$, define the submatrix A_x as the $n \times n$ matrix made by repeating the ith row of A x_i times. The target distribution of BosonSampling is dependent on the permanents of these submatrices. On input A, a column orthonormal complex matrix, the string $x \in \mathbb{N}^m$ should be sampled probability $Per(A_x)/x_0!x_1! \dots x_m!$.

This finishes the definitions necessary for this thesis. It is also noteworthy to explore connections between these classes, a quick representation is given in Figure 1.2. Of particular note to this thesis are the questions of $\mathsf{BQP} \stackrel{?}{=} \mathsf{BPP}$ or $\mathsf{FBQP} \stackrel{?}{=} \mathsf{FBPP}$, which ask whether quantum computers can solve decision problems or functional/search problems that classical

probabilistic computers cannot.

1.3.4 Proof Relativisation

In chapter 2 we will note that our particular style of proof contains insight beyond it's literal results. Namely, our proofs overcome the so called 'relativisation barrier' [16], and as such, they may serve as a starting point for further advancements in complexity theory. While a full description of the numerous barriers in complexity theory is outside the scope of this thesis, we can provide a brief introduction to the relativisation barrier.

The relativisation barrier is a concept used by complexity theorists to determine if a certain style of proof can be used to establish a result, based on whether that result holds (or fails to hold) relative to various oracles. To illustrate, suppose we are trying to prove two complexity classes are not equal, e.g. P and NP. If we know there exists an oracle A such that $\mathsf{P}^{\mathsf{A}} = \mathsf{NP}^{\mathsf{A}}$ and a second oracle B such that $\mathsf{P}^{\mathsf{B}} \neq \mathsf{NP}^{\mathsf{B}}$, then any proof of $\mathsf{P} \neq \mathsf{NP}$ must somehow not function when the classes are made relative to A [16].

For a number of results, we know the result does not hold relative to all oracles, thus we know the proof of the result must be 'non relativising'. This makes proof techniques which 'cross the relativisation barrier' very exciting for proving non relativising results.

1.4 Research questions

It is useful to frame this thesis with a number of research questions.

Research question 1
What is the strongest theoretical basis for the claim "In polynomial time quantum computers can perform computations that classical computers cannot"?

The first chapter to address this question is Chapter 2, which approaches from a complexity theoretic angle. Particularly, we see what unexpected complexity theory collapses might occur if FBQP is equal to FBPP. This chapter finds if $\mathsf{FBQP} = \mathsf{FBPP}$ then the polynomial hierarchy collapses to the second order. The strongest known collapse for any implementable quantum class. Chapter 3 asks subquestions to this research question, first asking if quantum computers can outperform classical on the task of generating advice (in the sense of $\mathsf{P/poly}$), then asking if quantum computers can outperform classical computers when receiving advice. Chapter

3 demonstrates the importance of these questions to the questions of $\mathsf{BQP} \stackrel{?}{=} \mathsf{P/poly}$ and $\mathsf{BQP} \stackrel{?}{=} \mathsf{BPP/samp}$. The answers to these subquestions naturally lead to the next research question.

Research question 2
If polynomial-time quantum computers can give better advice than polynomial-time classical computers, can we find such an advice-generating algorithm?

This research question is the topic of Chapter 6, which describes a machine learning algorithm which uses the quantum computer to find a good set of weights (the advice in this case), but which can then be deployed purely on a classical computer for any input. Indeed, we show that this algorithm captures all other surrogate models and is in some sense universal for this type of problem.

The machine learning model developed in Chapter 6 is naturally more suited to modern quantum computers because it does not require the use of an expensive quantum computer for each evaluation. This is a useful property to have and leads us to ask if we can further reduce the burden of quantum machine learning by reducing the requirements on the size of our quantum computer.

Research question 3
Do there exist methods to reduce the number of qubits required to run a given machine learning algorithm?

Chapter 4 first provides evidence that for some circuits this may be possible. It develops a quantum machine learning algorithm which can naturally simulate larger quantum circuits than those which are being used to deploy the quantum algorithm, it does this while also presenting a path to reduce the runtime of this simulation when such an option is available. Numerical evidence is then used to show that there exist examples where this algorithm is indeed capable of reducing qubit counts, by learning the output of a random quantum circuit using circuits of half the size. However, Chapter 5 shows that not all circuits can be reduced by this (or any other) method. It shows that there exist circuits such that the ability to remove even one qubit efficiently (lowering the number of qubits needed by even one) would imply $\mathsf{BQP} = \mathsf{BPP}$.