

## **QUANTUM SUPREMACY IN END-TO-END INTELLIGENT IT PT. III. QUANTUM SOFTWARE ENGINEERING – QUANTUM APPROXIMATE OPTIMIZATION ALGORITHM ON SMALL QUANTUM PROCESSORS**

**Ivancova Olga<sup>1</sup>, Korenkov Vladimir<sup>2</sup>, Tyatyushkina Olga<sup>3</sup>,  
Ulyanov Sergey<sup>4</sup>, Fukuda Toshio<sup>5</sup>**

<sup>1</sup>Senior researcher;

Joint institute for nuclear researches, Laboratory of Information Technologies;  
Dubna State University,  
Institute of system analysis and management;  
141980, Dubna, Moscow reg., Universitetskaya str., 19;  
e-mail: o\_ivancova@mail.ru.

<sup>2</sup>Doctor of Technical Science, professor;

Joint institute for nuclear researches, Laboratory of Information Technologies;  
Deputy Director of the Laboratory;  
141980, Moscow reg., Dubna, Joliot-Curie, 6;  
Dubna State University,  
Institute of system analysis and management;  
141980, Dubna, Moscow reg., Universitetskaya str., 19;  
e-mail: korenkov@cv.jinr.ru.

<sup>3</sup>Ph D, Associate professor;

Dubna State University,  
Institute of the system analysis and management;  
141980, Dubna, Moscow reg., Universitetskaya str., 19;  
e-mail: tyatyushkina@mail.ru.

<sup>4</sup>Doctor of Science in Physics and Mathematics, professor;

Joint institute for nuclear researches, Laboratory of Information Technologies;  
141980, Moscow reg., Dubna, Joliot-Curie, 6;  
Dubna State University,  
Institute of system analysis and management;  
141980, Dubna, Moscow reg., Universitetskaya str., 19;  
e-mail: ulyanovsv@mail.ru.

<sup>5</sup>PhD, professor; president IEEE;

Nagoya University,  
Dept. of Micro System, Dept. of Mechanics- Informatics;  
Furo-cho, Chikusa-ku, Nagoya, Japan;  
e- mail: fukuda@mein.nagoya u.ac.jp.

*Principles and methodologies of quantum algorithmic gate-based design on small quantum computer described. The possibilities of quantum algorithmic gates simulation on classical computers discussed. A new approach to a circuit implementation design of quantum algorithm gates for fast quantum massive parallel computing presented. SW & HW support sophisticated smart toolkit of supercomputing accelerator of quantum algorithm simulation on small quantum programmable computer algorithm gate (that can program in SW to implement arbitrary quantum algorithms by executing any sequence of universal quantum logic gates) described.*

**Keywords:** quantum algorithm, small quantum computer, quantum computation intelligence, quantum programming.

# КВАНТОВОЕ ПРЕВОСХОДСТВО В СКВОЗНЫХ ИНТЕЛЛЕКТУАЛЬНЫХ ИТ

## Ч. 3: КВАНТОВАЯ ПРОГРАММНАЯ ИНЖЕНЕРИЯ – КВАНТОВЫЕ АЛГОРИТМЫ ПРИБЛИЖЕННОЙ АППРОКСИМАЦИИ НА МАЛЫХ КВАНТОВЫХ ПРОЦЕССОРАХ

**Иванцова Ольга Владимировна<sup>1</sup>, Кореньков Владимир Васильевич<sup>2</sup>,  
Тятюшкина Ольга Юрьевна<sup>3</sup>, Ульянов Сергей Викторович<sup>4</sup>, Фукуда Тошио<sup>5</sup>**

<sup>1</sup>Старший преподаватель;  
ГБОУ ВО МО «Университет Дубна»,  
Институт системного анализа и управления;  
141980, Московская обл., г. Дубна, ул. Университетская, 19;  
e-mail: o\_ivancova@mail.ru.

<sup>2</sup>Доктор технических наук, профессор, директор;  
Объединенный институт ядерных исследований,  
Лаборатория информационных технологий;  
141980, Московская обл., г. Дубна, ул. Жолио-Кюри, 6;  
ГБОУ ВО МО «Университет Дубна»,  
Институт системного анализа и управления;  
141980, Московская обл., г. Дубна, ул. Университетская, 19;  
e-mail: korenkov@cv.jinr.ru.

<sup>3</sup>Кандидат технических наук, доцент;  
ГБОУ ВО МО «Университет Дубна»,  
Институт системного анализа и управления;  
141980, Московская обл., г. Дубна, ул. Университетская, 19;  
e-mail: tyatyushkina@mail.ru.

<sup>4</sup>Доктор физико-математических наук, профессор;  
ГБОУ ВО МО «Университет Дубна»,  
Институт системного анализа и управления;  
141980, Московская обл., г. Дубна, ул. Университетская, 19;  
Объединенный институт ядерных исследований,  
Лаборатория информационных технологий;  
141980, Московская обл., г. Дубна, ул. Жолио-Кюри, 6;  
e-mail: ulyanovsv@mail.ru.

<sup>5</sup>Доктор наук, профессор;  
Нагоя университет,  
Факультет микросистем, механики и информатики;  
Япония, Нагоя, Фуру-чо, Чикуса-ку;  
e-mail: fukuda@mei.nagoya-u.ac.jp.

Описаны принципы и методологии проектирования квантовых алгоритмических ячеек на малых квантовых компьютерах. Обсуждаются возможности моделирования квантовых алгоритмических ячеек на классических компьютерах. Описаны приложения схмотехнических решений квантовых ячеек. Представлен новый подход к реализации схмотехнических решений квантовых алгоритмов для быстрых квантовых параллельных массивных вычислений. Основное внимание уделено разработке метода проектирования операторов быстрых квантовых алгоритмов, таких как суперпозиция, запутывание и интерференция, которые в общем случае являются трудоемкими операциями из-за количества выполненных продуктов. Программно-алгоритмическая платформа поддерживает сложный интеллектуальный инструментальный ускоритель моделирования квантового алгоритма на малом квантовом компьютере, на котором реализуются квантовые алгоритмы путем выполнения последовательности универсальных логических элементов реализации квантовой логики.

**Ключевые слова:** квантовые алгоритмы, малый квантовый компьютер, сильный вычислительный интеллект, квантовое программирование.

## Introduction

Quantum computers are still in an early stage of development, and experimental quantum processors are getting to support up to a few dozen of qubits. That number is growing, and support for a large number of qubits is just a matter of time. Where we speak of quantum computers in this paper, we mean gate-based quantum computers and not quantum annealers. Software tools for programming gate-based models quantum computers are also in an early stage of development, where the basics, however, were already set 20 years ago. Currently, these software tools are mainly supporting a set of low-level quantum instructions, embedded in a classical (digital) programming language.

They are adequate for running rudimentary quantum applications that can already run on only a few qubits. Quantum computing should now grow towards a phase where the development of quantum software must get more emphasis. This is comparable with the sixties of the previous century, when programming tools like Fortran and Algol brought a much higher abstraction to digital computers than just assembly language.

Quantum Software Engineering toolkits for programming gate-based quantum computers are being developed by many parties. These tools should now grow towards a phase where they support quantum devices running realistic algorithms to outperform classical algorithms on digital computers. At this moment, they lack capabilities for generic gates, capabilities for quantum debugging and generic quantum libraries.

This paper gives a review on the functionalities needed for such software environments looking at the various layers of the software stack and at the interfaces for quantum cloud computing. Important that assume following: (1) quantum processors remain bulky devices for a long time (like digital computers in the beginning), (2) that real quantum applications will always be a mix of digital and quantum computing, where only a part of the problem is solved in a quantum manner, and (3) that we want to make this available for a larger public, that has limited knowledge about the underlying quantum layer. For this, it is required (1) a strict interface between remote quantum hardware, (2) local software that runs at the user side that can, (a) independent of the used local programming language, interact with the remote hardware and software, (b) independent of the used local programming language, make use of high level quantum algorithms in libraries, and (c) has profound quantum specific debugging capabilities where (small versions of) the desired algorithms can be analyzed, while stepping through the program in debug mode, up to the underlying quantum states using simulations.

The list of available tools is too long to be duplicated here, but a first impression can be gained by organizing them according to the used programming language.

Table 1 shows the programming languages that are being used for implementing quantum tools, and the number of each of these quantum tools.

*Table 1. Shortlist of languages being used for quantum software tools*

C / C++ based (>30)	Matlab / Octave based (12)
F# based (1)	Maxima based (2)
GUI based (>10)	NET based (4)
Java based (>15)	Online service (10)
JavaScript based (1)	Perl / PHP based (3)
Julia based (1)	Python based (>6)
Maple based (3)	Scheme/Haskell/LISP/ML based (8)
Mathematica based (8)	

What can be observed in this table is the wide range of different programming languages that has been used. There is no clear winner from that, simply because almost any classical programming language is suitable for this job, with each of these languages having its own advantages and disadvantages.

Another way to group the tools is taking a high-level view on their functionality. It recognized the following functional groups: (a) quantum compilers, (b) quantum function libraries, and (c) back-end quantum simulators. It is aware that some tools may not fully fit into one particular group and these groups may overlap, but it is helpful to identify several high-level functionalities. We will now discuss the characteristics of each of these groups one by one.

**Quantum compilers.** The word compiler refers in this context to a tool that translates an entire program from higher level statements into some lower level instructions (e.g., binary or assembly). The execution of

that program is started thereafter. The purpose of a compiler is to generate low level quantum instructions (in assembly or as native code), from a mix of low- and high-level quantum statements, of classical control statements and loops, of organized code in reusable routines, etc.

Quantum tools that present themselves as quantum compilers are mainly modifications from an existing (classical) compiler. The result is a changed syntax to have it extended with quantum specific instructions. The present extensions are mainly to offer a syntax for elementary instructions at the level of quantum assembly code. Table 2 offers a few examples of those tools.

*Table 2. Examples of quantum compiling tools*

Name	Remarks
Scaffold or ScafCC	C-alike, updated version of CLANG. Compiles to LLVM & QASM [10].
Liquid (Microsoft)	F#-alike, updated version of F# (called Q#), with an emulator on board [11].
Quipper	Haskel-alike [12]
QCL	mix of C and Pascal alike, detailed language specification with emulator on board [13].

Their main disadvantage is that creating a new language may break access to existing code libraries with classical algorithms. For instance, the development of quantum algorithms for outperforming classical machine learning algorithms will draw significant benefits if classical code libraries with, for example, neural network algorithms are well-accessible from the same software environment. Next to this, quantum compilers that introduce new languages, or break compatibility with existing languages lack in most cases powerful development and (quantum) debugging tools.

**Quantum function libraries.** The term function library refers in this context to the use of an existing and well-supported (classical) programming environment, where the quantum programmer can call quantum specific routines from a library. Their purpose is similar to that of compilers, with the difference that the generation of low-level quantum code occurs in run time. Examples of such (QLM, Atos), PyQuil (Rigetti), ProjectQ (ETH Zurich), and quantum tools are: Qiskit (IBM), Quantum Learning Machine OpenQL (TU Delft). The approach of using function libraries brings hardly any limitation, think of generating thousands of dedicated quantum instructions by a single call to a library function. For instance, a single call to a routine for a Quantum Fourier Transform or a matrix expression, that automatically generates hundreds of quantum instructions operating on tens of qubits. However, the present quantum function libraries have a strong focus on low-level quantum computing and may only be targeted at a specific quantum processor. Quantum programming with the present tools is mainly a matter of calling routines for generating individual low-level quantum instructions (like assembler). By calling several of them with different parameters, one can generate a sequence of quantum instructions to build a quantum circuit. The identification and handling of higher-level quantum instructions is still a topic of further research.

Some of these tools already offer powerful quantum debugging capabilities, like the generation of a drawing of the generated quantum circuit and access to a build-in quantum simulator. Such simulator can return intermediate results, like, for instance, a quantum state, that can never be obtained using a real quantum processor. Some of these tools also offer a graphical user interface, for positioning a few quantum gates into a quantum circuit to process a few qubits. That approach mainly serves an educational purpose for those who are setting their first steps in quantum computing. However, as soon as your quantum program grows in size, the use of scripting for calling quantum functions may become more convenient.

**Back-end simulators.** The term back-end simulator refers in this context to a tool that reads low level quantum assembly and executes them in a manner like a real quantum processor would do. As such, it becomes irrelevant for the quantum programmer in what language such tool is written since that tool is instructed directly via low level assembly instructions. The concepts of high-level instructions and function libraries are not applicable here and therefore we consider it as another group. Back-end simulators contain a very simple translator, to feed low level assembly instructions (usually stored in an ASCII file) to a build-in simulation engine. They are by definition dedicated to low-level functionality only and can simulate/emulate on a digital computer what a real quantum processor would have returned. These tools aim at simulating a real quantum processor as good as possible on a classical digital computer. They may even try to simulate the physical imperfections of a particular quantum processor as well. For instance, by adding some random mechanism

(noise) to mimic the loss of quantum coherence after executing more and more quantum instructions, or by deliberately accounting for topological limitations of a particular quantum processor. These tools may be offered as a stand-alone tool, or as part of a larger software environment (sometimes referred to as ‘virtual quantum machine’). The translation functionality they possess, may also be used to interface with a real quantum processor, but that is out of scope here. Examples of such quantum tools are (modules inside) QX (QuTech), QLM (Atos), QVM (Rigetti) and Quantum Experience (IBM).

These tools serve purposes other than quantum function libraries and quantum compilers do. They are valuable to study the quantum assembly language, to study how to deal with limitations in the quantum instruction set of the target machine, or to study quantum error correction methods against decoherence. They are also valuable to study the interfacing between a local software environment and a quantum processor somewhere in the cloud.

### *Desired software functionality*

The functionality described expect in quantum software, such that it supports implementation of more complex, hybrid digital and quantum algorithms and that it will enable a larger public that has limited knowledge about the underlying quantum layer to access quantum computing.

**Commonly available functionality.** In spite of all functional differences between the discussed tools, they all share a common functionality: the concept of quantum circuits for representing a set of quantum instructions via interconnected gates, and the concept of quantum assembler or Quantum Assembly Language (QASM) as a language for describing those circuits as a list of sequential quantum instructions. There are several of these quantum assembly languages, each with their own dialect (syntax), but from a pure conceptual point of view they all are roughly the same.

Table 3 shows an example of two different QASM dialects, both describing the same quantum circuit.

The first one shows the syntax used by QLM (from Atos), the second one shows the syntax used by QX (within Quantum Inspire). A QASM file is typically a sequence of individual QASM instructions, embedded between a header (with declarations) and a footer. Some QASM tools also allow for grouping those instructions into macros (like functions) to simplify the use of the same code multiple times. Each QASM instruction is build-up from (a) an instruction name, (b) optional parameters, and (c) a list of qubit identifiers on which this instruction should operate. Each instruction can change the contents of a ‘quantum register’ (via gates or measurements), and many of them in sequence define a quantum circuit. There is an apparent consensus on the naming of several of these gates, for instance, instruction names like H, X, Y, Z, but this consensus does not hold for all gates. This difference can be confusing, but is not a big issue when well defined. When these names are well specified by means of the corresponding matrix representation, the conversion from one QASM dialect into another is pretty straight forward. And when a particular gate is not available in one QASM, it can always be created in another QASM via a combination of a few other gates.

*Table 3. Example of two different dialects of quantum assembler, representing the same quantum circuit*

Atos-QASM	QX
BEGIN	
qubits 18	qubits 18
cbits 0	.ckt
H q[0]	H q[0]
H q[5]	H q[5]
CTRL(PH[1.570]) q[15],q[16]	CR q[15], q[16], 1.570
CTRL(PH[0.785]) q[14],q[16]	CR q[14], q[16], 0.785
CTRL(PH[0.392]) q[13],q[16]	CR q[13], q[16], 0.392
H q[15]	H q[15]
...	...
H q[16]	H q[16]
END	

Note that some QASM dialects start their counting of qubits from 0, while others start from 1. This is only a matter of convention and preference, mainly driven by the supporting language, and not a big issue either. In conclusion, one may say that it is relatively easy to translate one QASM dialect into another one. To reach the

desired level of quantum programming that supports the implementation of more complex, hybrid quantum algorithms and to enable a larger public that has limited knowledge about the underlying quantum layer to access quantum computing, we need a next level of software functionalities.

Examples of these functionalities are:

#### *Desired capabilities for generic gates*

- Define circuit libraries with generic gates, with an arbitrary number of qubits, and callable as if it was a single instruction. OpenQASM does support the concept of macros and can provide this functionality, but that concept is not available in all QASMs. The desired circuit library should generate, for instance, a Quantum Fourier Transform, or a circuit for modular exponentiation by one instruction/call for an arbitrary large number of qubits.
- Define circuit libraries with generic gates in terms of an unitary matrix or a matrix expression, while the tool translates that into circuits with only basic gates. This capability is currently a weak point for almost all tools. It may be available for one or two qubit gates, but the issues get problematic for more qubits.

#### *Desired capabilities for quantum debugging*

- The capabilities to let the software draw a circuit from a QASM specification to debug what has been specified. Tools like Liquid, QLM and ProjectQ give support for that, but it should be available on all tools of interest.
- The capability to read out the full vector (or full matrix) with complex numbers representing the present quantum state or circuit (also at intermediate points). This is impossible with a real quantum processor, and only possible with a simulator. However, it is a very powerful and essential debugging facility. Some simulation tools do support this, but there are very primitive solutions among them.
- The capability to analyse generic quantum states and gates (during simulation), while stepping through the program in debug mode, using sophisticated linear algebra tooling.
- The capability to visualize in an abstract manner relevant aspect of the (full) state vector or (full) matrix in case they are too large for a full numerical inspection. For instance, histograms, magnitude plots by means of colours, etc,

#### *Desired quantum libraries*

- Libraries that implement many quantum functions/circuits, callable from your quantum program. Many functions are well known from the literature, but inserting, for instance, a Quantum Fourier Transform operating on an arbitrary number of qubits (in arbitrary order) should be as simple as inserting a single qubit X gate.
- Running quantum programs from a programming environment with good access to classic libraries with legacy solutions for the quantum application under study. This may mean (a) a language that is different from the implementation language of the quantum tools, and/or (b) software that prepares the quantum application locally (for debugging reasons) for running it on a remote quantum processor (via a well-supported API, Application Programming Interface), and/or (c) software that uses a different operating system from the one used by the quantum processor.

These desired functionalities are examples only, and the list is not complete. They have currently a strong focus on (quantum) debugging capability, flexibility, and libraries. The present tools may implement fragments of this list, but we have not found a tool that can support them all in a convenient manner.

### ***Desired software environment***

If we know what functionality we need in the software to enter the next level of quantum computing, we should define where and how this functionality should be implemented and where the separation between local and cloud can be placed.



**Layered software stack.** To define where and how the functionality should be implemented, we use a full software stack with different layers, as shown in Fig. 1, covering functionality from quantum applications down to quantum processors. The concept in Fig. 1 is shown.

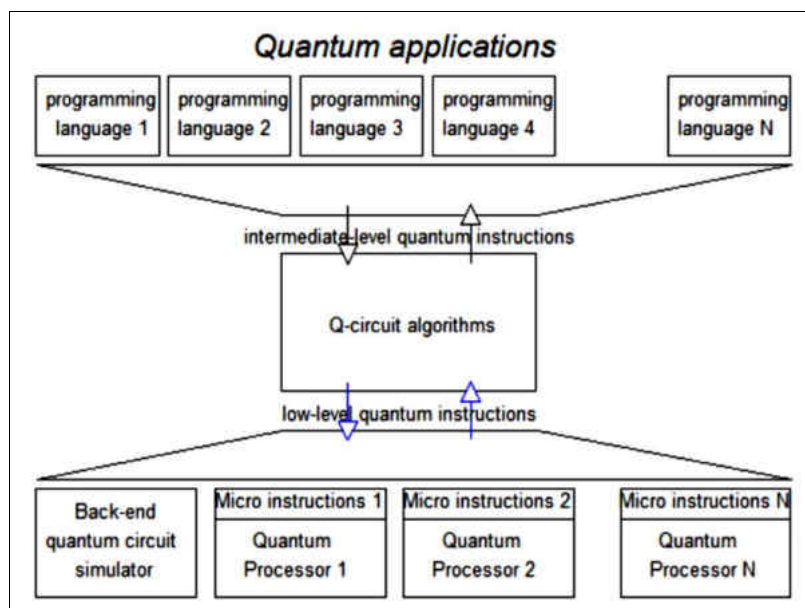


Figure 1. Software stack for quantum tooling with different layers

Put more emphasis on the desired functionality within the intermediate levels.

1) *Functionalities of bottom Layers:* The very bottom of the low-level layers are reserved for the quantum processors, each equipped with dedicated hardware for controlling it via (binary) micro instructions. For instance, to change the quantum state of a quantum processor via dedicated pulses. A software layer directly above this hardware allows for translating a sequence of low-level quantum instructions (e.g., assembly or binary code) into dedicated pulses for the hardware. This software layer should hide the hardware difference among different quantum processors as much as possible, in order to program them with uniform instructions that are more or less hardware-independent. In practice, very different quantum processors will be implemented, based on different physical principles, and each with their own micro instruction sets. The quantum programmer should not be bothered by that. These instructions are still very low-level, and the use of some QASM dialect is the most obvious choice here. These instructions allow for defining quantum circuits with basic gates, where the word basic refers to a set of predefined gates operating on 1 or 2 qubits only. These instructions may be restricted to the (hardware) instruction range of the target quantum processor, and may also account for topological (hardware) limitations. Since quantum processors are still under development, the use of a back-end circuit simulator also has its place in the lower layers. Their aim is to simulate a real quantum processor as fast as possible, with as many qubits as the used digital computer platform can handle, and to simulate/emulate all its imperfections and, if applicable, all topology limitations as good as possible.

2) *Functionalities of intermediate Layers:* A more intermediate level (the Q-circuit algorithms box in Fig. 2) gives the quantum programmer access to all kinds of quantum algorithms, in a uniform manner, ideally independent from the used quantum hardware and programming languages. It is, for instance, equipped with all kinds of algorithm libraries (examples can be found in the next section) and quantum debugging capabilities to design the quantum-specific parts of applications. These instructions are at least capable of defining arbitrary circuits with generic gates. This means within this context that they can operate on an arbitrary number of qubits, far beyond the instruction set of the quantum processor, and they may even be specified via (unitary) matrices or high-level expressions. The translation from quantum circuit with a few generic gates into circuits with many basic gates is the proper place to perform gate and qubit optimization. The best results can be achieved when this translation is partly guided by control parameters representing some of the hardware limitations of the target quantum processor (hardware aware). These control parameters are set only once for a particular target quantum processor, and preferably invisible in the instructions with generic gates (hardware

agnostic). As such, this translation is an important intermediate step in quantum programming, applicable to both quantum compilers and tools based on function libraries.

Figure 2 provides an example of a quantum circuit with generic gates.

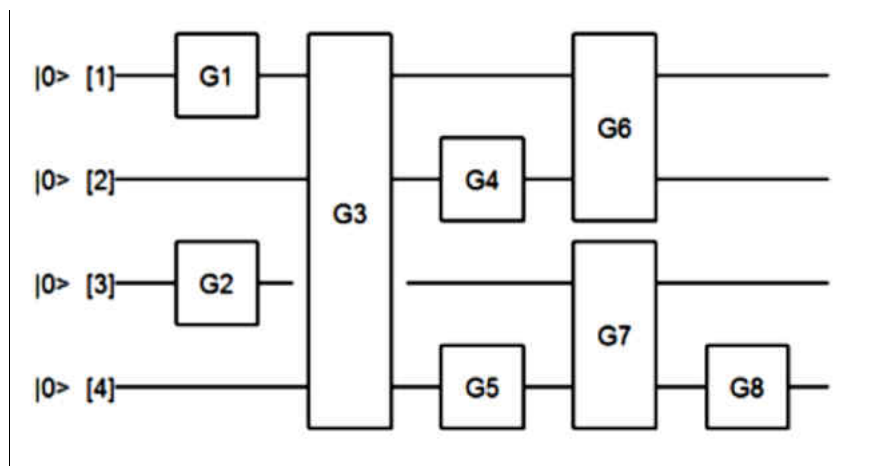


Figure 2. Example of a quantum circuit with generic gates

At a first glance, generic may look the same as basic gates, but in this case the gates can also be black boxes operating on many qubits simultaneously specified as matrices, expressions, or standard functionalities. One may still consider the supported instructions as rather low-level, but the distinction between low and intermediate level brings a significant advantage. If quantum computing is offered via the cloud, for running quantum applications from all over the world, then the interface between the lower and intermediate layer is a natural interface for the cloud-based quantum computer. And the use of one or more QASM dialects is a natural component in the interfacing between these layers. It means that software in the intermediate layers may fully run on a local computer, while software in the lower layer should typically run on a remote quantum host.

3) *Functionalities of higher layers*: The higher layers (above the Q-circuit algorithms box in Fig. 1) are typically reserved for classical programming environments that are extended with quantum capabilities for solving dedicated subproblems. Here the new group of users, with limited knowledge of underlying quantum techniques should be able to play around. It would be a waste of effort if each programming environment has to develop its own collection of libraries with quantum-specific algorithms. Therefore, it is far more efficient to equip them only with language-specific interfaces, wrapped around common quantum libraries and debugging capabilities from the intermediate layer. Quantum applications will most likely be a hybrid mix of classical programming concepts and quantum-specific algorithms. This means that the classical programming languages call a quantum algorithm only when needed for solving particular

subproblems. These classical programming languages should offer the following capabilities:

- Good access to classical software libraries, with dedicated algorithms for the problem area. Think of libraries for artificial intelligence applications. But think also of access to quantum circuit libraries with dedicated quantum-specific algorithms like quantum solutions for dealing with decoherence errors or for decomposing a large number into its prime factors.
- Good access to quantum debugging capabilities, also for the quantum specific aspects. Think of inspecting quantum states and matrices of quantum circuits via a build-in simulator and drawing quantum circuits from instructions. These debugging capabilities should easily interact with the higher layers, all in a very interactive and flexible manner.

As such, the universal programming language for everybody does not exist, and therefore the best solution for that is that the intermediate layers offer access to quantum-specific libraries for any programming language of interest. It supports many quantum circuit algorithms as well as quantum-specific debugging capabilities. The use of languages with build-in support for linear algebra expressions, that are also available as interpreter, give the user powerful extra capabilities for quantum debugging. These tools allow for inspecting and manipulating intermediate results of circuit matrices and state vectors in a very interactive manner during simulation. Linear algebra languages like Matlab/Octave, and to some extent Python with Numpy, are



examples of languages offering the desired linear algebra capabilities in an interactive environment and offer access to a broad spectrum of classical code libraries.

**Separation between Local and Cloud.** It is assumed that quantum computers remain big installations with bulky refrigeration equipment for a long time. Commercial deployment of quantum computing will then mean a quantum computer hosted in a remote building, offering access via the cloud to many users all over the world. Today, experimental quantum computers already given access via the cloud, but mainly in a restricted manner; end users have to setup a remote terminal session with the hosting computer and they should develop and run everything on that host. This is quite inconvenient as it limits rapid interaction with local software, like exchanging intermediate results with local debugging software. Moreover, commercial users may not be willing to share their source code with the hosting organization. Exchange of low-level code (binary or assembly) gives then a similar protection as is common today for distributing programs as a binary executable. In that case, end users develop, test and debug on a small scale (locally) at an intermediate level, and then send low level quantum instructions to a remote host in order to run at full quantum speed and size. The most convenient way of implementing that is therefore not by opening remote terminal sessions, but by accessing the remote quantum computer via an API. An API allows the user to program his (quantum) application in a language that differs from the programming language being used by the remote host. The intermediate layers will then send QASM-like quantum instructions to a remote host, while the end-user experiences it as if it runs locally.

Figure 3 illustrates this interaction model, where the interface between local and cloud is situated between the intermediate and lower layers.

The intermediate layers are equipped with all kinds of libraries for quantum specific calculations, as well as debugging capabilities via a local quantum simulator. These libraries generate the required low level QASM instructions and can forward them through a language-independent interface to the lower layers running in the cloud.

To get an idea of the type of algorithms that can be implemented via libraries, we will discuss a few examples: libraries that convert a mathematical expression into a quantum circuit, libraries that decompose a generic gate or generic matrix description of such gates into circuits with basic gates and libraries that convert arithmetic calculations into a quantum circuit.

*Example: Quantum arithmetic libraries.* Several quantum applications make use of algorithms where discrete numbers are represented by distinct quantum states. For instance, algorithms that make use of modular additions, modular multiplications or modular exponentiations. In those cases, it is not obvious what the most efficient way is to implement these on many qubits.

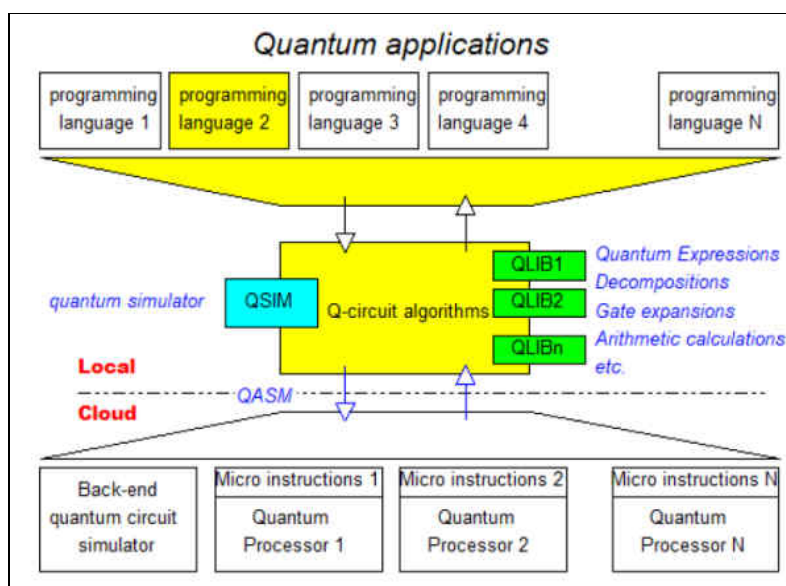


Figure 3. Quantum software stack overview

We will show three example circuits of how to calculate something ‘simple’ like the modular increment of a discrete number encoded in 4 qubits. The first one (Fig. 4 (a)) can be found in any textbook, looks quite

simply, however, requires gates with many control inputs. The second one (Fig. 4 (b)) with Peres gates is also known, looks more complicated, but requires only single qubits gates and single controlled qubit gates.

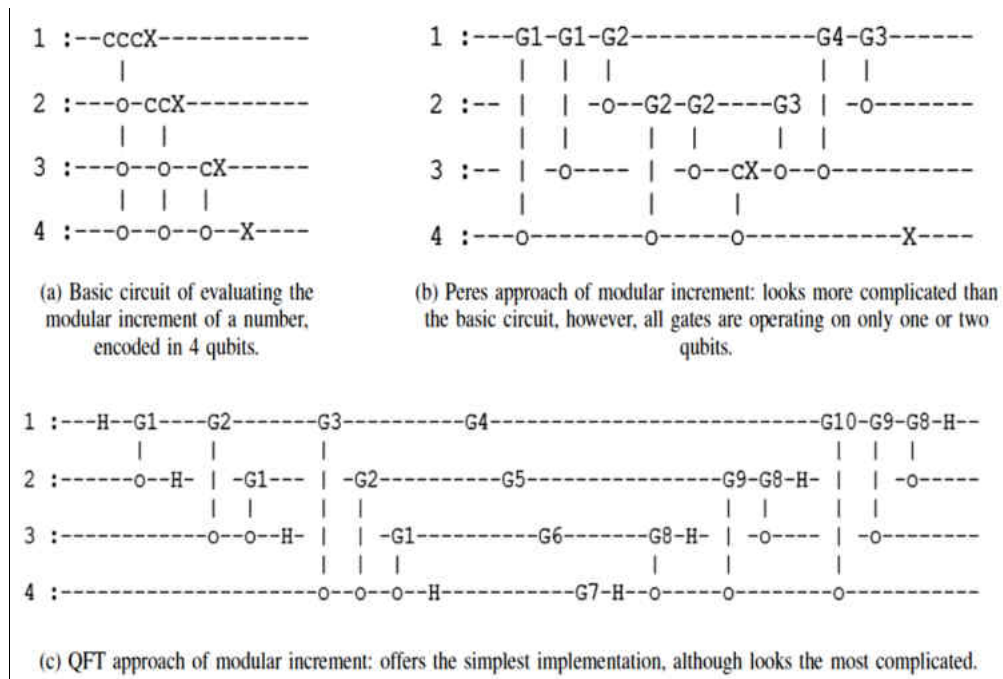


Figure 4. Three examples of evaluating the modular increment of a number, encoded in 4 qubits

The used gates are explained in Table 4. The third example (Fig. 4c) using quantum Fourier transforms, however, appeared to be the simplest one of these three, in the sense that only single qubit gates and controlled phases are used. This may illustrate that generating an algorithm with the most efficient circuit is not obvious even for a very simple problem. There are many more of these modular arithmetic calculations, each of them with multiple implementations. Their details are out of scope here, but it may again illustrate the value of bringing all these implementations together into a well-organized “quantum arithmetic library”.

*Remark.* The list of useful libraries is unlimited. Think of circuits to generate a Quantum Fourier Transform, or to step through a quantum random walk. The same applies for the best way to deal with error corrections, to deal with the topology limitations of a particular quantum computer, or to build standard circuits by using only the native gates of a particular quantum computer (those that can be made with a single pulse). Note that implementations of basic gates like X, Y, Z, may require more than one pulse (this depends on the physical implementation being used). Existing tools have some of those algorithms implemented.

Table 4. Explanation of used gates in Figs 4c and 4b.

Peres approach	QFT approach
$G1 = c([1 + q4, 1 - q4; 1 - q4, 1 + q4]/2)$	$G1 = cR(\pi/2)$
$G2 = c([1 + q2, 1 - q2; 1 - q2, 1 + q2]/2)$	$G2 = cR(\pi/4)$
$G3 = G2'$ (conjugated transpose of G2)	$G3 = cR(\pi/8)$
$G4 = G1'$ (conjugated transpose of G1)	$G4 = Rs(\pi/8)$
	$G5 = Rs(\pi/4)$
	$G6 = Rs(\pi/2)$
	$G7 = Rs(\pi)$
	$G8 = cR(-\pi/2)$
	$G9 = cR(-\pi/4)$
	$G10 = cR(-\pi/8)$
where	where
$q2 = (-j)$	$Rs(\phi) = [1, 0; 0, e^{(j*\phi)}]$
$q4 = \sqrt{(-j)}$	$cR(\phi) = c(Rs(\phi))$
$c([g_{11}, g_{12}; g_{21}, g_{22}]) =$	
$[1, 0, 0, 0;$	
$0, 1, 0, 0;$	
$0, 0, g_{11}, g_{12};$	
$0, 0, g_{21}, g_{22}]$	

However, a common library that can be used by different quantum tools is currently only in an early phase of development.

## Gate Level Quantum Software Platforms

An overview of various quantum computers and the software needed to connect to them is shown in Fig. 5.

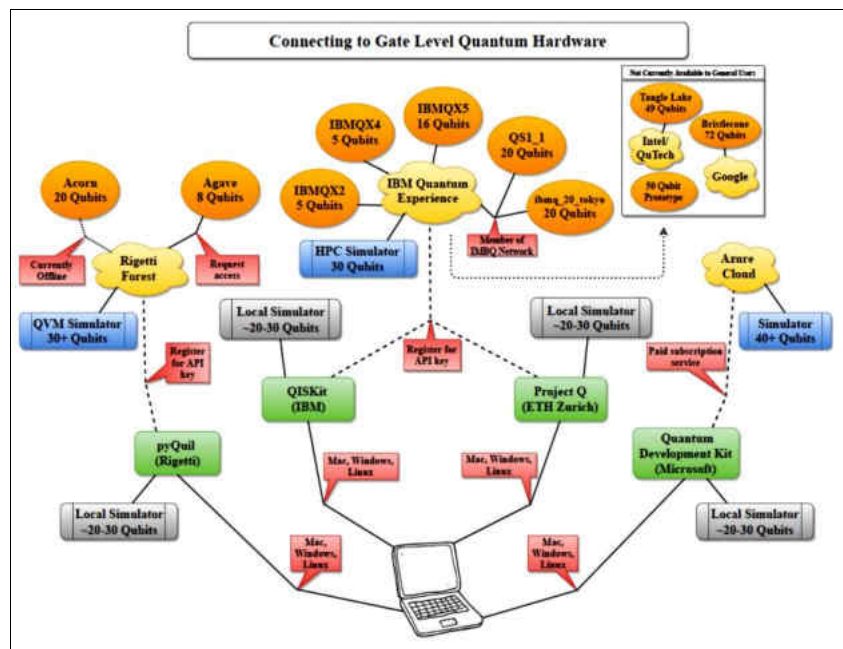


Figure 5. A schematic diagram showing the paths to connecting a personal computer to a usable gate-level quantum computer [5]

Starting from the personal computer (bottom center), nodes in green shows software that can be installed on the user's personal computer. Grey nodes show simulators run locally (i.e., on the user's computer). Dashed lines show API/cloud connections to company resources shown in yellow clouds. Quantum simulators and usable quantum computers provided by these cloud resources are shown in blue and gold, respectively. Red boxes show requirements along the way. For example, to connect to Rigetti Forest and use the Agave 8 qubit quantum computer, one must download and install pyQuil (available on macOS, Windows, and Linux), register on Rigetti's website to get an API key, then request access to the device via an online form.

*Notes:* (i) Rigetti's Quantum Virtual Machine requires an upgrade for more than 30 qubits, (ii) local simulators depend on the user's computer so numbers given are approximates, and (iii) the grey box shows quantum computers that have been announced but are not currently available to general users.

As it currently stands, these four software platforms allow one to connect to four different quantum computers one by Rigetti, an 8 qubit quantum computer which can be connected to via pyQuil; and three by IBM, the largest openly available being 16 qubits, which can be connected to via QISKit or ProjectQ. There is also a fourth 20 qubit quantum computer by IBM, but this device is only available to members of the IBM Q Network, a collection of companies, universities, and national laboratories interested in and investing in quantum computing. Also shown in Fig. 5 are quantum computers by companies like Google, IBM, and Intel which have been announced but are not currently available to general users.

An example of the same quantum circuit compiled by both of these platforms is shown in Fig. 6. Here, with pyQuil compile to the Agave specifications and with QISKit compile to the IBMQX5 specifications.

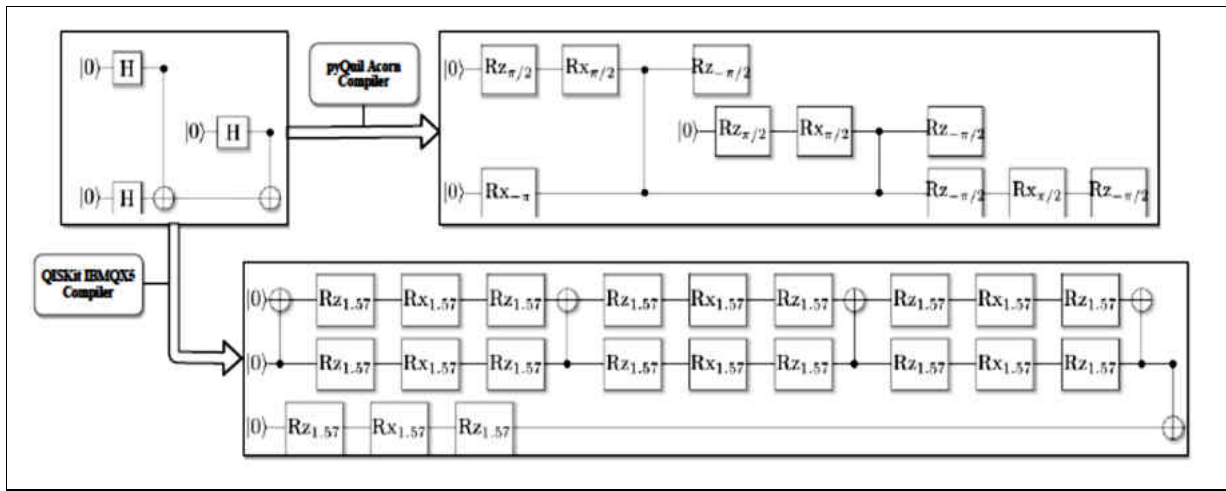


Figure 6. An example of a quantum circuit (top left) compiled by pyQuil for Rigetti's 8 qubit Agave processor (top right), and the same circuit compiled by QISKit for IBM's 16 qubit IBMQX5. The qubits used on Agave are 0, 1, and 2, and the qubits used on IBMQX5 are 0, 1, and 2. Note that neither compiler can directly implement a Hadamard gate  $H$  but produces these via products of rotation gates  $R_x$  and  $R_z$ . A CNOT gate can be implement on IBMQX5, but not on Agave [Here, pyQuil must express CNOT in terms of Controlled-Z and rotations. These circuits were made with ProjectQ]

As can be seen, QISKit produces a longer circuit (i.e., has greater depth) than pyQuil. It is not appropriate to claim one compiler is superior because of this example, however. Circuits that are in the language IBMQX5 understands would naturally produce a shorter depth circuit than pyQuil, and vice versa. It is known that any quantum circuit (unitary matrix) can be decomposed into a sequence of one and two qubit gates, but in general this takes exponentially many gates. It is currently a question of significant interest to find an optimal compiler for a given topology.

ProjectQ comes with a high-performance C++ simulator that performed the best in local testing. The maximum size circuit we were able to successfully simulate was 28 qubits, which took just under ten minutes (569.71 seconds) with a circuit of depth 20. For the complete performance and testing, see Fig. 7.

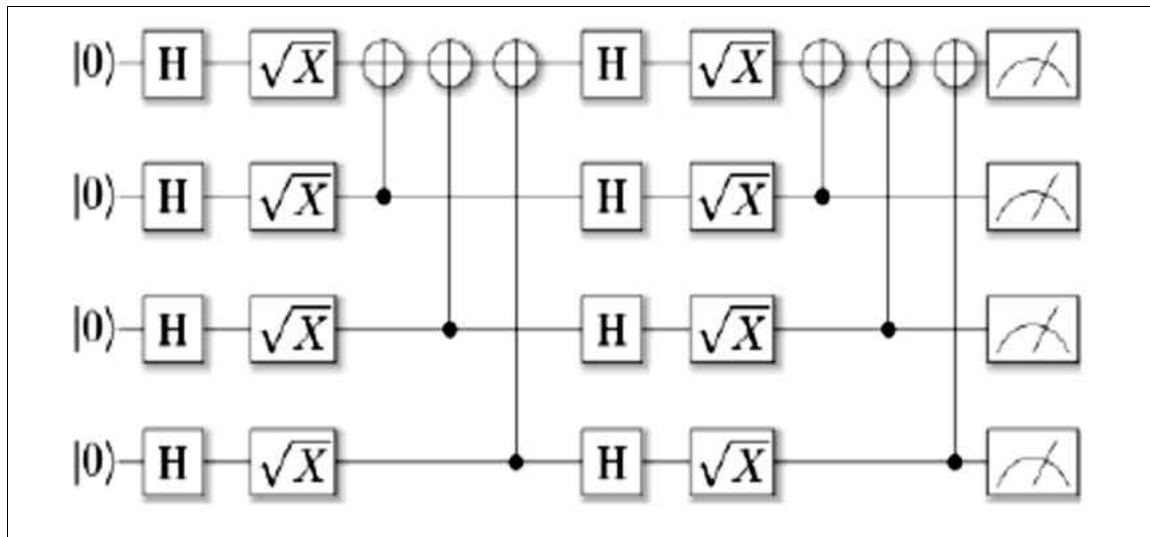


Figure 7. The circuit used for testing the ProjectQ C++ simulator and QISKit local state vector simulator, shown here on four qubits. In the actual testing, the pattern of Hadamard gates,  $\sqrt{X}$  gates, then the sequence of CNOT gates defines one level in the circuit. This pattern is repeated until the desired depth is reached. This image was produced using ProjectQ

We introduce the fundamentals and current challenges when using quantum computers during the *Noisy Intermediate-Scale Quantum* (NISQ). Unitary operations are represented as gates, combined with qubits and measurements they form a quantum circuit. Such quantum circuits are gate-based representations of quantum algorithms. The number of gate collections to be executed sequentially is defined as the depth of a quantum circuit. Within such a collection, called layer, gates are performed in parallel. The number of qubits is defined as the width of the circuit. Both properties determine the required number of qubits and the stable execution time of a suitable quantum computer. Each quantum computer has a set of physically implemented gates. However, the sets of implemented gates differ from quantum computer to quantum computer. Thus, to create quantum circuits for specific problems, gates that are not implemented on the specific quantum computer must be realized by a combination of available gates. This is done by the hardware-specific transpiler of the vendor. Therefore, the transpiler maps the gates and qubits of the circuit to the gate sets and qubits of the regarded quantum computers. The resulting transpiled circuit may have a different depth and width than the general circuit. Especially the depth can differ greatly between different quantum computers. the limitation is that these qubits will still be error-prone because for the correction of such errors many more qubits are needed. Thus, these quantum computers are also called NISQ machines.

Selecting an appropriate quantum computer to execute a certain quantum algorithm not only depends on the mathematics of the algorithm itself but also on the physical requirements of its implementations.

**Software Development Kit (SDK)** Current implementations of quantum algorithms are tightly coupled to the SDKs they are developed with. Companies like IBM and Rigetti offer their proprietary SDKs for their quantum computers, called Qiskit and Forest5, respectively. There are also other SDKs that support the quantum computers of multiple vendors, e.g., ProjectQ [20] or XACC. Nonetheless, most of the SDKs only support quantum computers of a single vendor as backends. Furthermore, implementations are not interchangeable between different SDKs because of their different programming languages and syntax.

As a result, the majority of the developed implementations are only executable on a certain set of quantum computers provided by a specific vendor. approach for a NISQ Analyzer, which enables an automated analysis and selection of algorithm implementations and quantum hardware depending on the chosen quantum algorithm and specific input data. Fig. 8 depicts an overview of the approach.



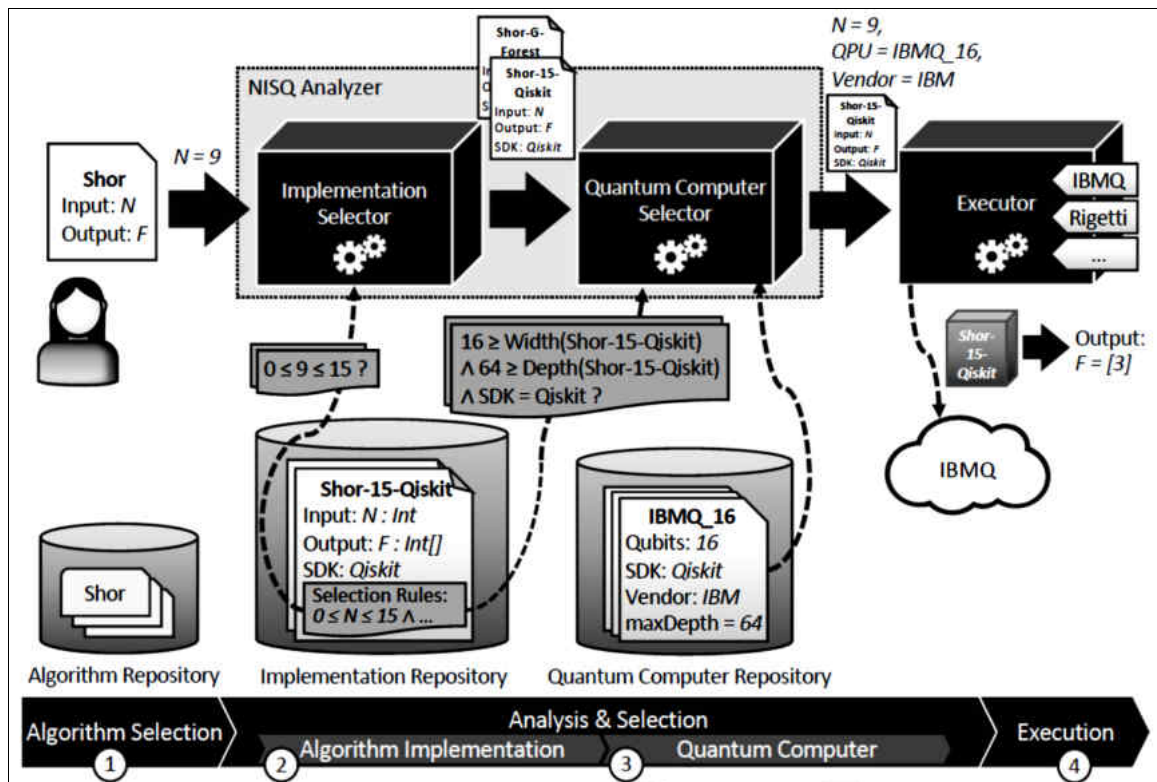


Figure 8. Approach for automated implementation and quantum hardware selection

The NISQ Analyzer analyzes and selects for a chosen quantum algorithm and specific input data (i) an appropriate algorithm implementation and (ii) a suitable quantum computer, the implementation can be executed on, by means of defined selection rules. Thereby, the width and depth of the algorithm implementations are dynamically determined using hardware-specific transpilers and compared with the properties of available quantum computers. Since quantum computers and implementations are tightly coupled to their SDKs, the compatibility between the SDK used for the implementation and the SDK supported by the quantum computer has to be considered. The selected implementation is then sent to the corresponding vendor of the selected quantum computer.

### Quantum-assisted quantum compiling

Currently available quantum computers are not large-scale but rather have been called noisy intermediate-scale quantum (NISQ) computers. A proof-of-principle demonstration of quantum supremacy with a NISQ device may be coming soon. Nevertheless, demonstrating the practical utility of NISQ computers appears to be a more difficult task.

Quantum computing theorists can contribute to the utility of NISQ devices by developing software. This software would aim to adapt textbook quantum algorithms (e.g., for factoring or quantum simulation) to NISQ constraints. NISQ constraints include: (1) limited numbers of qubits, (2) limited connectivity between qubits, (3) restricted (hardware-specific) gate alphabets, and (4) limited circuit depth due to noise. Algorithms adapted to these constraints will likely look dramatically different from their textbook counterparts. These constraints have increased the importance of the field of quantum compiling. In classical computing, a compiler is a program that converts instructions into assembly language so that they can be read and executed by a computer. Similarly, a quantum compiler would take a high-level algorithm and convert it into a lower-level form that could be executed on a NISQ device.

**Quantum-assisted quantum compiling (QAQC**, pronounced “Quack”). The goal of QAQC is to compile a (possibly unknown) target unitary to a trainable quantum gate sequence. A key feature of QAQC is the fact that the cost is computed directly on the quantum computer. This leads to an exponential speedup (in the number of qubits involved in the gate sequence) over classical methods to compute the cost, since classical simulation of quantum dynamics is exponentially slower than quantum simulation. Consequently, one should



be able to optimally compile larger-scale gate sequences using QAQC, whereas classical approaches to optimal quantum compiling will be limited to smaller gate sequences.

Define a cost function for QAQC that satisfies the following criteria:

1. It is faithful (vanishing if and only if the compilation is exact);
2. It is efficient to compute on a quantum computer;
3. It has an operational meaning;
4. It scales well with the size of the problem.

As a proof-of-principle, we implement QAQC on both IBM's and Rigetti's quantum computers, and compile various one-qubit gates to the native gate alphabets used by these hardware.

Figure 9 illustrates four potential applications of QAQC. Suppose that there exists a quantum algorithm to perform some task, but its associated gate sequence is longer than desired. As shown in Fig. 9 (a), it is possible to use QAQC to shorten the gate sequence by accounting for the NISQ constraints of the specific computer.

This depth compression goes beyond the capabilities of classical compilers. As a simple example, consider the quantum Fourier transform on  $n$  qubits. Its textbook algorithm is written in terms of Hadamard gates and controlled-rotation gates, which may need to be compiled into the native gate alphabet. The number of gates in the textbook algorithm is  $O(n^2)$ , so one could use a classical compiler to locally compile each gate. But this could lead to a sub-optimal depth since the compilation starts from the textbook structure. In contrast, QAQC is unbiased with respect to the structure of the gate sequence, taking a holistic approach to compiling as opposed to a local one. Hence, in principle, it can learn the optimal gate sequence for given hardware. Note that classical compilers cannot take this holistic approach for large  $n$  due to the exponential scaling of the matrix representations of the gates.

Alternatively, consider the problem of simulating the dynamics of a given quantum system with an unknown Hamiltonian  $H$  (via  $e^{-iHt}$ ) on a quantum computer. This problem black-box called because by simulating the black-box, i.e., the unitary  $e^{-iHt}$ , we are “uploading” the unitary onto the quantum computer. This scenario is depicted in Fig. 9(A) (b). QAQC could be used to convert an analog black-box unitary into a gate sequence on a digital quantum computer.

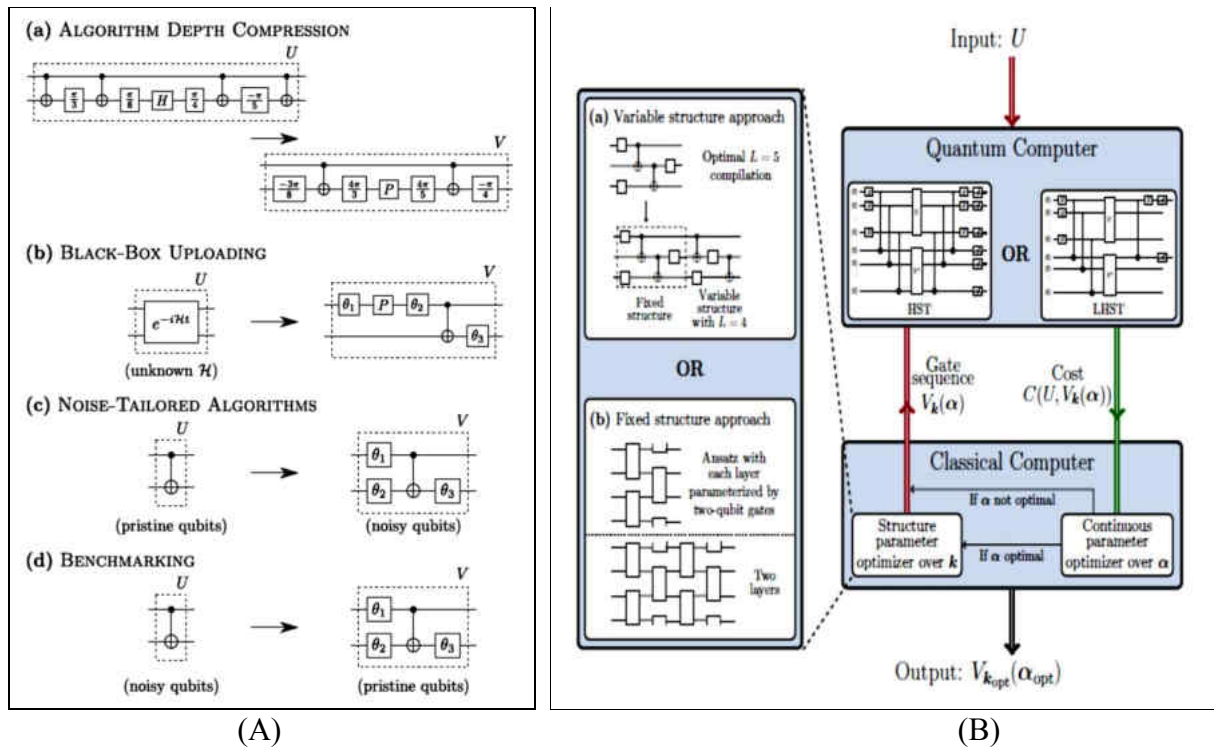


Figure 9 (A): Potential applications of QAQC. (a) Compressing the depth of a given gate sequence  $U$  to a shorter depth gate sequence  $V$  in terms of native hardware gates. (b) Uploading a black-box unitary. The

*black box could be an analog unitary  $U = e^{-iHt}$ , for an unknown Hamiltonian  $H$ , that one wishes to convert into a gate sequence to be run on a gate-based quantum computer. (c) Training algorithms in the presence of noise to learn noise-resilient algorithms (e.g., via gates that counteract the noise). Here, the unitary  $U$  is performed on high-quality, pristine qubits and  $V$  is performed on noisy ones. (d) Benchmarking a quantum computer by compiling a unitary  $U$  on noisy qubits and learning the gate sequence  $V$  on high-quality qubits [6]*

*Figure 9 (B): Outline of variational hybrid quantum-classical algorithm, in which optimize over gate structures and continuous gate parameters in order to perform QAQC for a given input unitary  $U$ . We take two approaches towards structure optimization: (a) For small problem sizes, allow the gate structure to vary for a given gate sequence length  $L$ , which in general leads to an approximate compilation of  $U$ . To obtain a better approximate compilation, the best structure obtained can be concatenated with a new sequence of a possibly different length, whose structure can vary. For each iteration of the continuous parameter optimization, calculate the cost using the Hilbert-Schmidt Test (HST); (b) For large problem sizes, fix the gate structure using an ansatz consisting of layers of two-qubit gates. By increasing the number of layers, can obtain better approximate compilations of  $U$ . For each iteration of the continuous parameter optimization, we calculate the cost using the Local Hilbert-Schmidt Test (LHST)*

Consider Fig. 9 (A) (c). Here, the goal is to implement a CNOT gate on two noisy qubits. Due to the noise, to actually implement a true CNOT, one has to physically implement a dressed CNOT, i.e., a CNOT surrounded by one-qubit unitaries. QAQC can be used to learn the parameters in these one-qubit unitaries. By choosing the target unitary  $U$  to be a CNOT on a pristine (i.e., noiseless) pair of qubits, it is possible to learn the unitary  $V$  that needs to be applied to the noisy qubits in order to effectively implement a CNOT. We call this application noise-tailored algorithms, since the learned algorithms are robust to the noise process on the noisy qubits.

Figure 9 (A) (d) depicts the opposite process, which is benchmarking. Here, the unitary  $U$  acts on a noisy set of qubits, and the goal is to determine what the equivalent unitary  $V$  would be if it were implemented on a pristine set of qubits. This essentially corresponds to learning the noise model, i.e., benchmarking the noisy qubits.

Quantum-classical hybrid strategy to perform the optimization is illustrated in Fig. 9 (B).

## Applications

First consider variational quantum eigensolvers and QAOA algorithms, which can be treated essentially identically using described techniques. Then consider the training of quantum autoencoders, which requires a slightly different formalism. The gradient descent algorithms can be applied to these problems by reducing such problems to a probability maximization problem. For each application our quantum gradient computation algorithm yields a quadratic speedup in terms of the dimension.

**Variational quantum eigensolvers.** Variational quantum eigensolvers (VQE) and QAOA have been favored methods for providing low-depth quantum algorithms for solving important problems in quantum simulation and optimization. Current quantum computers are limited by decoherence, hence the option to solve optimization problems using very short circuits can be enticing even if such algorithms are polynomially more expensive than alternative strategies that could possibly require long gate sequences. Since these methods are typically envisioned as being appropriate only for low-depth applications, comparably less attention is paid to the question of what their complexity would be, if they were executed on a fault-tolerant quantum computer. In this section, we consider the case that these algorithms are in fact implemented on a fault-tolerant quantum computer and show that the gradient computation step in these algorithms can be performed quadratically faster compared to the earlier approaches that were tailored for pre-fault-tolerant applications. VQEs are widely used to estimate the eigenvalue corresponding to some eigenstate of a Hamiltonian. The idea behind these approaches is to begin with an efficiently parameterizable ansatz to the eigenstate. For the example of ground state energy estimation, the ansatz state is often taken to be a unitary coupled cluster expansion. The terms in that unitary coupled cluster expansion are then varied to provide the lowest energy for the ground state. For excited states a similar argument can be applied, but minimizing a free energy rather than ground state energy is the most natural approach.

For simplicity, let us focus on the simplest (and most common) example of ground state estimation. Consider a Hamiltonian of the form  $H = \sum_j a_j U_j$  where  $U_j$  is a unitary matrix,  $a_j > 0$  and  $\sum_j a_j = 1$ . This assumption can be made without loss of generality by renormalizing the Hamiltonian and absorbing signs into the unitary matrix. Let the state  $|\psi(x)\rangle$  for  $x \in \mathbb{R}^d$  be the variational state prepared by the Prep. and Tuned circuits in Fig. A1.25b. The objective function is then to estimate  $x_{\text{opt}} = \arg \min_x \left( \langle \psi(x) | \sum_j a_j U_j | \psi(x) \rangle \right)$  which is real-valued because  $H$  is Hermitian. In order to translate this problem to one that it can handle using the gradient descent algorithm, and construct a verifier circuit that given  $|\psi(x)\rangle$  sets an ancilla qubit to 1 with probability  $p = (1 + \langle \psi(x) | H | \psi(x) \rangle) / 2$ . This is possible since  $\|H\| \leq 1$  due to the assumption that  $\sum_j a_j = 1$ . This motivates the definition of new input oracles used for implementing the Hamiltonian:

$$\text{prepare } W : |0\rangle \mapsto \sum_j \sqrt{a_j} |j\rangle, \text{ select } H := \sum_j |j\rangle\langle j| \otimes U_j.$$

Use the unitaries to define and compute the query complexity of performing a single variational step in a VQE algorithm (Fig. 10).

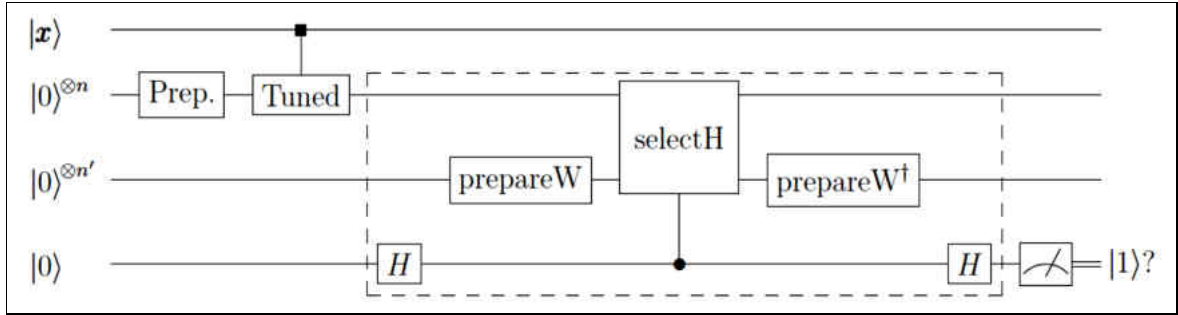


Figure 10. Circuit for converting ground state energy to a probability for VQE

The dashed box denotes the verifier circuit,  $V$ , in Fig. 9A,b which corresponds here to the Hadamard test circuit. Probability of measuring 1 is  $\frac{1}{2} - \frac{1}{2} \langle \psi(x) | H | \psi(x) \rangle$ . Note here Prep. circuit is the identity gate, which is kept in the circuit only for clarity. Note that in this context, the final prepare  $W^\dagger$  can be omitted.

Let on Fig. 10 operator  $\text{Tuned}(x) = \prod_{j=1}^d e^{-iH_j x_j}$  for  $\|H\| = 1$  and  $x \in \mathbb{R}^d$  and  $\text{Prep} = I$ . If  $H = \sum_{j=1}^M a_j H_j$  for unitary  $H_j$  with  $a_j \geq 0$  and  $\sum_j a_j = 1$  then the number of queries to prepare  $W$ , select  $H$  and Tunned needed to output a qubit string  $|y\rangle$  such that  $|\nabla \langle \psi(x) | H | \psi(x) \rangle - y| \leq \varepsilon$  with probability at least  $2/3$  is  $O(\sqrt{d} / \varepsilon)$ .

There are many flavors of the quantum approximate optimization algorithm (QAOA). The core idea of the algorithm is to consider a parametrized family of states such  $|\psi(x)\rangle = \prod_{j=1}^d e^{-x_j H_j} |0\rangle$ . The aim is to modify the state in such a way as to maximize an objective function. In particular, if we let  $O$  be a Hermitian operator corresponding to the objective function then we wish to find  $x$  such that  $\langle \psi(x) | H | \psi(x) \rangle$  is maximized. For example, in the case of combinatorial optimization problems the objective function is usually expressed as the number of satisfied clauses:  $O = \sum_{\alpha=1}^m C_\alpha$ , where  $C_\alpha$  is 1 if and only if the  $\alpha^{\text{th}}$  clause is satisfied and 0

otherwise. Such clauses can be expressed as sums of tensor products of Pauli operators, which allows us to express them as Hermitian operators. Thus, from the perspective of our algorithm, QAOA looks exactly like variational quantum eigensolvers except that the parameterization chosen for the state may be significantly different from that chosen for variational quantum eigensolvers.

**Quantum auto-encoders.** Classically, one application of neural networks is *auto-encoders*, which are networks that encode information about a data set into a low-dimensional representation. Auto-encoding was first introduced by Rumelhart et al. Informally, the goal of an auto-encoding circuit is the following: suppose we are given a set of high-dimensional vectors, we would like to learn a representation of the vectors hopefully of low dimension, so that computations on the original data set can be “approximately” carried out by working only with the low-dimensional vectors. More precisely the problem in auto-encoding is: Given  $K < N$  and  $m$  data vectors  $\{\nu_1, \dots, \nu_m\} \subseteq \mathbb{R}^N$ , find an encoding map  $\mathcal{E} : \mathbb{R}^N \rightarrow \mathbb{R}^K$  and decoding map  $\mathcal{D} : \mathbb{R}^K \rightarrow \mathbb{R}^N$  such that the average squared distortion  $\|\nu_i - (\mathcal{D} \circ \mathcal{E})(\nu_i)\|^2$  is minimized:

$$\min_{\mathcal{E}, \mathcal{D}} \sum_{i \in [m]} \frac{\|\nu_i - (\mathcal{D} \circ \mathcal{E})(\nu_i)\|^2}{m}$$

Given that classical auto-encoders are ‘work-horses’ of classical machine learning, it is also natural to consider a quantum variant of this paradigm. Very recently such quantum auto-encoding schemes have been proposed by Wan Kwak et al. and independently by Romero et al. Inspired by their work it is provided a slightly generalized description of quantum autoencoders by ‘quantizing’ auto-encoders the following way: we replace the data vectors  $\nu_i$  by quantum states  $\rho_i$  and define the maps  $\mathcal{E}, \mathcal{D}$  as quantum channels transforming states back and forth between the Hilbert spaces  $\mathcal{H}$  and  $\mathcal{H}'$ . A natural generalization of squared distortion for quantum states  $\rho, \sigma$  that we consider is  $1 - F^2(\rho, \sigma)$  giving us the following minimization

problem  $\min_{\mathcal{E}, \mathcal{D}} \sum_{i \in [m]} \frac{1 - F^2(\rho_i, (\mathcal{D} \circ \mathcal{E})(\rho_i))}{m}$ . Since  $F^2(|\psi\rangle\langle\psi|, \sigma) = \langle\psi|\sigma|\psi\rangle$  in the special case when the

input states are pure states  $\rho_i = |\psi_i\rangle\langle\psi_i|$ , the above minimization problem is equivalent to the maximization

problem  $\min_{\mathcal{E}, \mathcal{D}} \sum_{i \in [N]} \frac{\langle\psi_i|[(\mathcal{D} \circ \mathcal{E})(|\psi_i\rangle\langle\psi_i|)]|\psi_i\rangle}{m}$ . Observe that  $\langle\psi|[(\mathcal{D} \circ \mathcal{E})(|\psi\rangle\langle\psi|)]|\psi\rangle$  is the probability

of finding the output state  $(\mathcal{D} \circ \mathcal{E})(|\psi\rangle\langle\psi|)$  in state  $|\psi\rangle$  after performing the projective measurement  $\{|\psi\rangle\langle\psi|, I - |\psi\rangle\langle\psi|\}$ .

Thus, it can think about this as maximizing the probability of recovering the initial pure state after encoding and decoding, which is a natural measure of the quality of the quantum auto-encoding procedure.

**Training quantum auto-encoders.** Let us describe a way to perform this optimization problem in the special case when the input states are  $n$ -qubit pure states and they are mapped to  $k$ -qubit states, i.e.,  $\mathcal{H}$  is the Hilbert space of  $n$  qubits and  $\mathcal{H}'$  is the Hilbert space of  $k < n$  qubits. The gradient computation algorithm can speed up solving the described optimization problem.

By adding a linear amount of ancilla qubits we can represent the encoding and decoding channels by unitaries, which makes the minimization conceptually simpler. Indeed by Stinespring’s dilation theorem it know that any quantum channel  $\mathcal{E}$  that maps  $n$  qubit states to  $k$  qubit states can be constructed by adding  $2k$  qubits initialized in  $|\vec{0}\rangle$  state, then acting with a unitary  $U_{\mathcal{E}}$  on the extended space and then tracing out  $k + n$  qubits. Applying this result to both  $\mathcal{E}$  and  $\mathcal{D}$  results in a unitary circuit representing the generic encoding/decoding procedure, see Fig. 11. (This upper bound on the required number of ancilla qubits for  $\mathcal{D}$  becomes  $2n$ .)

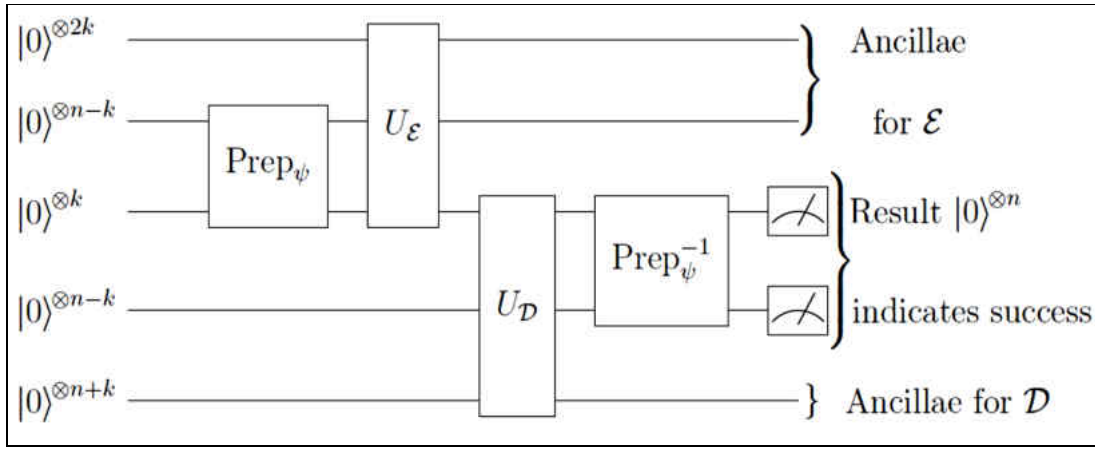


Figure 11. A unitary quantum auto-encoding circuit: For the input  $|\psi\rangle$ , the circuit prepares  $|\psi\rangle$ , applies a purified version of the channels  $\mathcal{E}, \mathcal{D}$  and finally checks by a measurement whether the decoded state is  $|\psi\rangle$

In order to solve the maximization problem we could just introduce a parametrization of the unitaries  $U_{\mathcal{E}}$ ,  $U_{\mathcal{D}}$  and search for the optimal parameters using gradient descent. Unfortunately, a complete parametrization of the unitaries requires exponentially many parameters, which is prohibitive. However, analogously to, e.g., classical machine learning practices, one could hope that a well-structured circuit can achieve close to optimal performance using only a polynomial number of parameters. If the circuits  $U_{\mathcal{E}}$ ,  $U_{\mathcal{D}}$  are parametrized nicely, then we can use our gradient computation algorithm to speed up optimization.

We can do the whole optimization using stochastic gradient descent, so that in each step we only need to consider the effect of the circuit on a single pure state. Or if we have more quantum resources available, we can directly evaluate the full gradient by preparing a uniform superposition over all input vectors. In this case the state preparation unitary  $\text{Prep} = \sum_{i=1}^m |i\rangle\langle i| \otimes \text{Prep}_{\psi_i}$  is a controlled unitary, which controlled on index  $i$  would prepare  $\psi_i$ .

Graphically this type of control is represented by a small black square in contrast to the small black circle used for denoting simple controlled unitaries. (See the full quantum circuit in Fig. 12.)

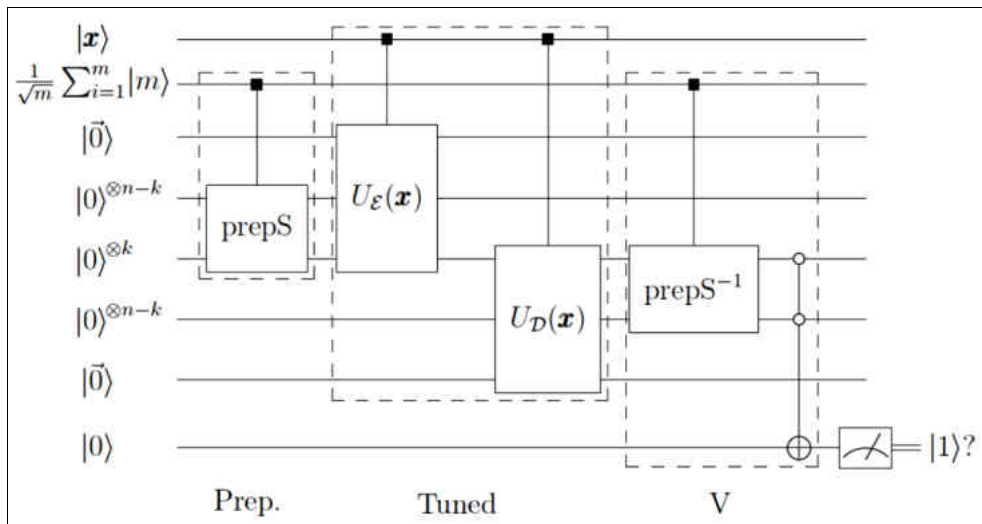


Figure 12. Quantum circuit which outputs 1 with probability equal to the objective function. The structure of the circuit fits the generic model of quantum optimization circuits, therefore can use the gradient computation methods to speed-up its optimization

Finally, note that in some application it might be desirable to ask for a coherent encoding/decoding procedure, where all the ancilla qubits are returned to the  $|\dot{0}\rangle$  state. In this case one could define  $U_p = U_\varepsilon^{-1}$  and optimize the probability of measuring  $|\dot{0}\rangle$  on the ancilla qubits after applying  $U_\varepsilon$ .

The main applied domains of QAG - approach: (i) Computer science; (ii) Artificial intelligence (AI) industrial applications; (iii) Fundamental intelligent informatics and general system theory; and (iv) Intelligent robust control system design in next two books are considered.

These domains be in deep interdependence. And included quantum algorithms design (quantum walk models, one-way computing and quantum adiabatic computing) in many applied quantum software engineering: quantum machine learning; quantum game models and decision-making processes in information uncertainty; quantum pattern / face recognition; quantum error correction codes; quantum-inspired neural network structures design (for quantum learning) and quantum-inspired genetic search algorithms (for quantum global optimization) etc.

It is the background of quantum soft computing for robust KB design of intelligent control system and robust quantum control (quantum soft computing applications), etc.

We described briefly any important applications of QAG simulation system (as examples, quantum games, quantum random search walk algorithms and control decision-making processes in classical and quantum situations of information uncertainty) and its developed tools in AI-systems design. Main ideas and peculiarities of Quantum Soft Computing tools as a new paradigm of computational intelligence and simulation processes briefly are considered. Applied Quantum Soft Computing toolkit (as a quantum computational toolkit and background for robust intelligent control design technology) discussed.

Some quantum algorithms already exist that show a theoretical speedup compared to the best known classical algorithms, the implementation and execution of these algorithms come with several challenges. The input data determines, e.g., the required number of qubits and gates of a quantum algorithm. An algorithm implementation also depends on the used Software Development Kit which restricts the set of usable quantum computers. Because of the limited capabilities of current quantum computers, choosing an appropriate one to execute a certain implementation for a given input is a difficult challenge that requires immense mathematical knowledge about the implemented quantum algorithm as well as technical knowledge about the used Software Development Kits. Thus, it exist a roadmap for the automated analysis and selection of implementations of a certain quantum algorithm and appropriate quantum computers that can execute the selected implementation with the given input data.

There are several challenges regarding the execution of quantum algorithms, which are considered in the following. There exists a multitude of different implementations for quantum algorithms that are only applicable to certain input data, e.g., in terms of the number of qubits required for its encoding, which we refer to as input size. These implementations differ from each other in various aspects, e.g., the required number of qubits and operation. Platforms that provide connectivity to real quantum devices must necessarily have a means of translating a given circuit into operations the computer can understand. This process is known as compilation, or more verbosely quantum circuit compilation/quantum compilation. Each computer has a basis set of gates and a given connectivity - it is the compiler's job to take in a given circuit and return an equivalent circuit obeying the basis set and connectivity requirements. For example, Qiskit and Rigetti, for these are the platforms with real quantum computers. An example of the same quantum circuit compiled by both of these platforms is shown in Fig. 6.

Here, with pyQuil compile to the Agave specifications and with Qiskit compile to the IBMQX5 specifications. As can be seen, Qiskit produces a longer circuit (i.e., has greater depth) than pyQuil. It is not appropriate to claim one compiler is superior because of this example, however. Circuits that are in the language IBMQX5 understands would naturally produce a shorter depth circuit than pyQuil, and vice versa. It is known that any quantum circuit (unitary matrix) can be decomposed into a sequence of one and two qubit gates, but in general this takes exponentially many gates. It is currently a question of significant interest to find an optimal compiler for a given topology.



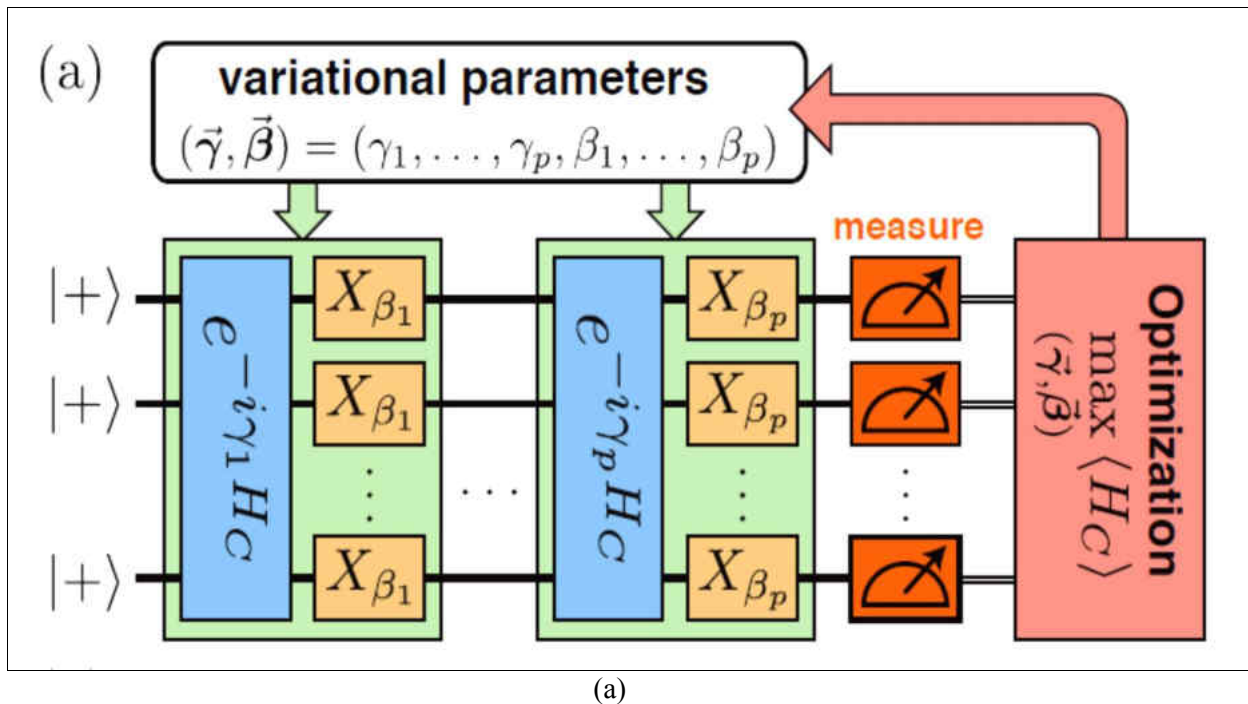
## Quantum optimization algorithms via faster quantum gradient computation

One of the technical challenges in applying the gradient computation procedure for quantum optimization problems and quantum deep machine learning is the need to convert between a probability oracle (which is common in quantum optimization procedures) and a phase oracle (which is common in quantum algorithms) of the objective function  $f$ . Efficient subroutines to perform this delicate interconversion between the two types of oracles incurring only a logarithmic overhead provided, which might be of independent interest. Finally, using these tools the runtime of prior approaches for training quantum auto-encoders, variational quantum eigensolvers (VQE), and quantum approximate optimization algorithms (QAOA) improved.

The Quantum Approximate Optimization Algorithm (QAOA) is a hybrid quantum-classical variational algorithm designed to tackle combinatorial optimization problems. Despite its promise for near-term quantum applications, not much is currently understood about QAOA's performance beyond its lowest-depth variant. An essential but missing ingredient for understanding and deploying QAOA is a constructive approach to carry out the outer-loop classical optimization. A critical problem to assess QAOA at intermediate depths where one may hope for a quantum computational advantage. One major hurdle lies in the difficulty to efficiently optimize in the non-convex, high-dimensional parameter landscape. Without constructive approaches to perform the parameter optimization, any potential advantages of the hybrid algorithms could be lost.

The QAOA is a quantum algorithm recently introduced to tackle these combinatorial optimization problems. To encode the problem, the classical objective function can be converted to a quantum problem Hamiltonian by promoting each binary variable to a quantum spin. For  $p$ -level QAOA, which is visualized in Fig. 13 (a), the quantum processor initialized in the state  $|+\rangle^{\otimes N}$ , and then apply the problem Hamiltonian  $H_C$

and a mixing Hamiltonian  $H_B = \sum_{j=1}^N \sigma_j^x$  alternately with controlled durations to generate a variational wavefunction  $|\psi_p(\vec{\gamma}, \vec{\beta})\rangle = e^{-i\beta_p H_B} e^{-i\gamma_p H_C} \dots e^{-i\beta_1 H_B} e^{-i\gamma_1 H_C} |+\rangle^{\otimes N}$ , which is parameterized by  $2p$  variational parameters  $\gamma_i$  and  $\beta_i$  ( $i=1, \dots, p$ ).



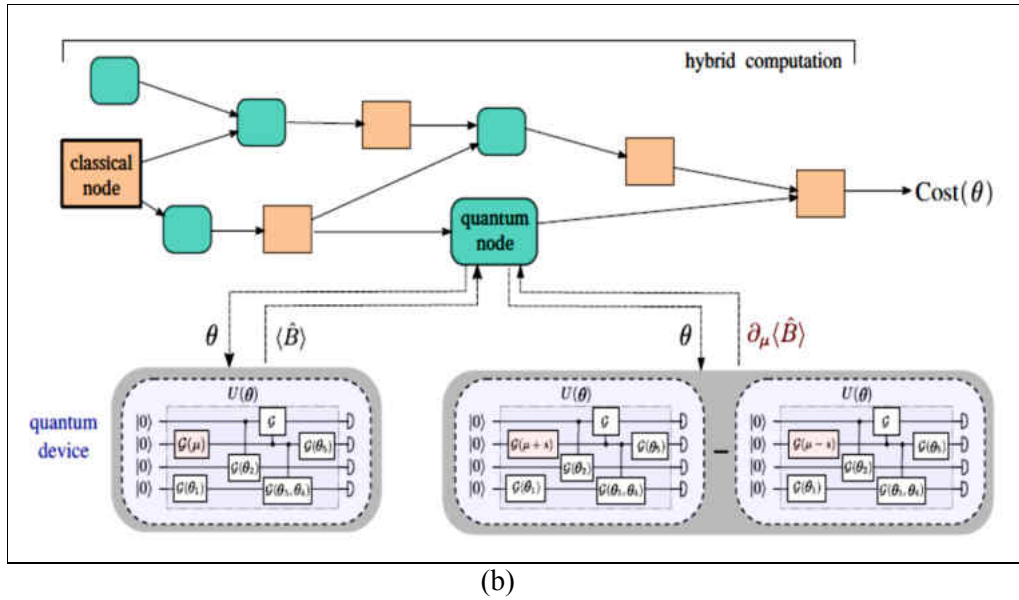


Figure 13. (a) Schematic of a  $p$ -level Quantum Approximation Optimization Algorithm [7, 8]

A quantum node, in which a variational quantum algorithm on Fig. 13,b is executed, can compute derivatives of its outputs with respect to gate parameters by running the original circuit twice, but with a shift in the parameter in question. The quantum algorithm or circuit consists of a gate sequence  $U(\theta)$  that depends on a set  $\theta$  of  $m$  real gate parameters, followed by the measurement of an observable  $\hat{B}$  and finding the expectation value of the measurement  $\hat{B}$  as a variational circuit. In the overall hybrid computation one can therefore understand a variational circuit as a function  $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$ , mapping the gate parameters to an expectation,  $f(\theta) := \langle \hat{B} \rangle = \langle 0 | U^\dagger(\theta) \hat{B} U(\theta) | 0 \rangle$ . While this abstract definition of a variational circuit is exact, its physical implementation on a quantum device runs the quantum algorithm several times and averages measurement samples to get an estimate of  $f(\theta)$ . Just like the variational circuit itself has an 'analytic' definition and a 'stochastic' implementation, the evaluation of these rules with finite runs on noisy hardware return estimates of the gradient.

On Fig. 13, a quantum circuit takes input  $|+\rangle^{\otimes N}$  and alternately applies  $e^{-i\gamma_i H_C}$  and  $X_{\beta_i} = e^{-i\beta_i \sigma_x}$ , and the final state is measured to obtain expectation value with respect to the objective function  $H_C$ . This is fed to a (classical) optimizer to find the best parameters  $(\vec{\gamma}, \vec{\beta})$  that maximizes  $\langle H_C \rangle$ . To better understand the mechanism of QAOA and make a meaningful comparison with QA, the QAOA parameters can be interpreted as a smooth annealing path. The sum of the variational parameters to be the total annealing time, i.e.,

$$T_p = \sum_{i=1}^p (|\gamma_i| + |\beta_i|). \text{ A smooth annealing path can be constructed from QAOA optimal parameters as}$$

$$H_{\text{QAQA}} = -[f(t)H_C + (1-f(t))H_B], \quad f\left(t_i = \sum_{j=1}^i (|\gamma_j^*| + |\beta_j^*|) - \frac{1}{2}(|\gamma_i^*| + |\beta_i^*|)\right) = \frac{\gamma_i^*}{|\gamma_i^*| + |\beta_i^*|},$$

where  $t_i$  is chosen to be at the mid-point of each time interval  $\gamma_i^* + \beta_i^*$ . With the boundary conditions  $f(t=0) = 0, f(t=T_p) = 1$  and linear interpolation at other intermediate time  $t$ , QAOA parameters can be converted to a well-defined annealing path. Apply this conversion to the QAOA optimal parameters at  $p = 40$  [Fig. 14 (e)]. With this annealing path, follow the instantaneous eigenstate population throughout the quantum annealing process [Fig. 14 (f)]. In contrast to adiabatic QA, the state population leaks out of the ground state and accumulates in the first excited state before the anticrossing point, where the gap is at its minimum. Using

a diabatic transition at the anti-crossing, the two states swap populations, and a large ground state population is obtained in the end.

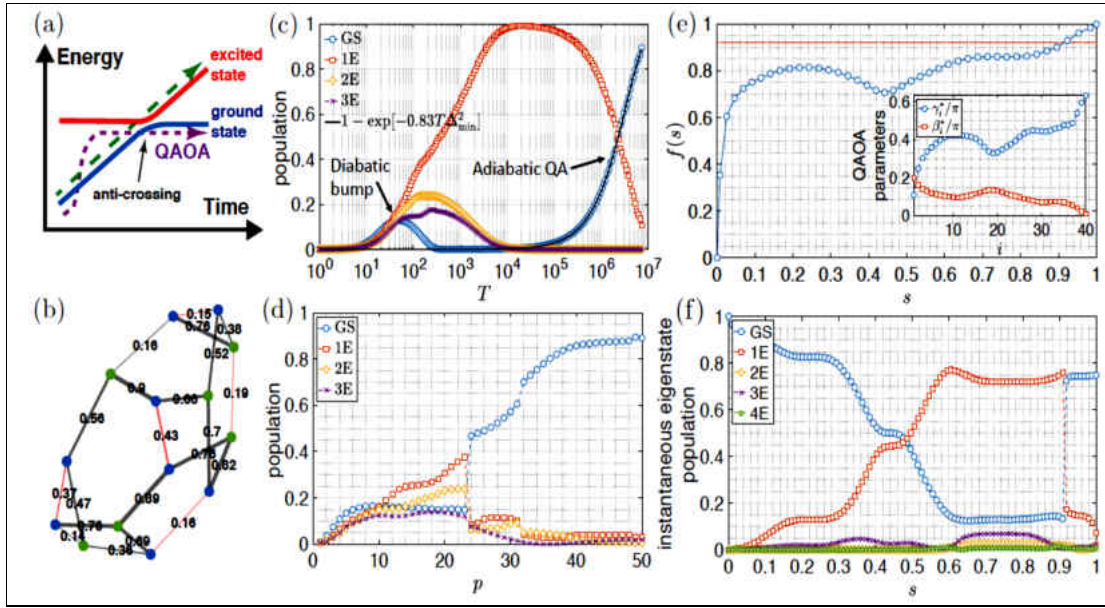


Figure 14. (a) A schematic of how QAOA and the interpolated annealing path can overcome the small minimum gap limitations via diabatic transitions (purple line). Naive adiabatic quantum annealing path leads to excited states passing through the anti-crossing point (green line). (b) An instance of weighted 3-regular graph that has a small minimum spectral gap along the quantum annealing path. The optimal MaxCut configuration is shown with two vertex colors, and the black (red) lines are the cut (uncut) edges. (c) Population in low excited states after the quantum annealing protocol with different total time  $T$ . The black solid line follows the Landau-Zener formula for the ground state population,

$p_{GS} = 1 - \exp(-cT\Delta_{\min}^2)$  where  $c$  is a fitting parameter. (d) Population in low excited states using QAOA at

different level  $p$ . The FOURIER heuristic strategy is used in the optimization. (e) Interpreting QAOA parameters (at  $p = 40$ ) as a smooth quantum annealing path, via linear interpolation. The annealing time

parameter  $s = t / T_p T_p =$ , where  $T_p = \sum_{i=1}^p (|\gamma_i^*| + |\beta_i^*|)$ . The red dotted line labels the location of anti-

crossing where the gap is at its minimum, at which point  $f(s) \approx 0.92$ . Inset shows the original QAOA

optimal parameters  $\gamma_i^*$  and  $\beta_i^*$  for  $p = 40$ . (f) Instantaneous eigenstate population under the annealing path given in (e). Note that the instantaneous ground state and first excited state swap at the anti-crossing point

Note that the final state population from the constructed annealing path differs slightly from those of QAOA, due to Trotterization and interpolation, but the underlying mechanism is the same, which is also responsible for the diabatic bump seen in Fig. 14 (c). In addition to the conversion tested a few other prescriptions to construct an annealing path from QAOA parameters, and qualitative features do not seem to change.

## Automatic differentiation of hybrid quantum-classical computations: Quantum software engineering

Near-term quantum devices require routines that are of shallow depth and robust against errors. The design paradigm of *hybrid algorithms* which integrate quantum and classical processing has therefore become increasingly important. Possibly the most well-known class of hybrid algorithms is that of *variational circuits*, which are parameter-dependent quantum circuits that can be optimized by a classical computer with regards to a given objective. Hybrid optimization with variational circuits opens up a number of new research avenues for near-term quantum with applications. There are various publicly accessible platforms to simulate quantum

algorithms or even run them on real quantum devices through a cloud service. However, even though some frameworks are designed with variational circuits, there is at this stage no unified tool for the hybrid optimization of quantum circuits across quantum platforms, treating all simulators and devices on the same footing.

SW PennyLane is a Python 3 software framework for optimization and machine learning of quantum and hybrid quantum-classical computations. The library provides a unified architecture for near-term quantum computing devices, supporting both qubit and continuous-variable paradigms. PennyLane's core feature is the ability to compute gradients of variational quantum circuits in a way that is compatible with classical techniques such as backpropagation. PennyLane thus extends the automatic differentiation algorithms common in optimization and machine learning to include quantum and hybrid computations. A plugin system makes the framework compatible with any gate-based quantum simulator or hardware. We provide plugins for Strawberry Fields, Rigetti Forest, Qiskit, Cirq, and ProjectQ, allowing PennyLane optimizations to be run on publicly accessible quantum devices provided by Rigetti and IBM Q. On the classical front, PennyLane interfaces with accelerated machine learning libraries such as TensorFlow, PyTorch, and autograd. PennyLane can be used for the optimization of variational quantum eigensolvers, quantum approximate optimization, quantum machine learning models, and many other applications.

In particular, PennyLane is an open-source Python 3 framework that facilitates the optimization of quantum and hybrid quantum-classical algorithms. It extends several seminal machine learning libraries — including *autograd*, *TensorFlow*, and *PyTorch* — to handle modules of quantum information processing. This can be used to optimize variational quantum circuits in applications such as *quantum approximate optimization* or *variational quantum eigensolvers*.

PennyLane can in principle be used with any gate-based quantum computing platform as a backend, including both qubit and continuous-variable architectures, and has a simple Python-based user interface.

Figure 15 shows a simple example that illustrates the core idea of the framework.

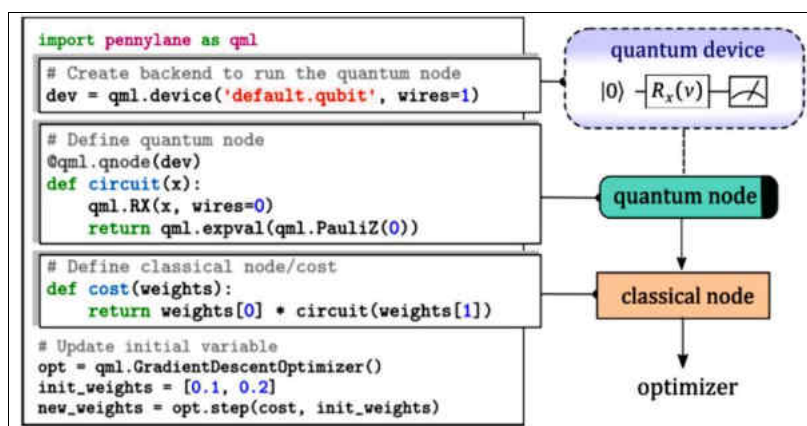


Figure 15. Basic example of a PennyLane program consisting of a quantum node followed by a classical node. The output of the classical node is the objective for optimization [9]

The user defines a quantum circuit in the function circuit connected to a device dev, as well as a “classical function” that calls circuit and computes a cost. The functions can be depicted as nodes in a directed acyclic computational graph that represents the flow of information in the computation. Each node may involve a number of input and output variables represented by the incoming and outgoing edges, respectively.

A Gradient DescentOptimizer is created that improves the initial candidate for these variables by one step, with the goal of decreasing the cost. PennyLane is able to automatically determine the gradients of all nodes - even if the computation is performed on quantum hardware - and can therefore compute the gradient of the final cost node with respect to any input variable. PennyLane is an open-source software project. The source code for PennyLane is available online on GitHub<sup>1</sup>, while comprehensive documentation and tutorials are available on PennyLane.

**Hybrid optimization** The goal of optimization in PennyLane is to find the minima of a cost function that quantifies the quality of a solution for a certain task. In hybrid quantum-classical optimization, the output of



the cost function is a result of both classical and quantum processing, or a *hybrid computation*. We call the processing submodules *classical* and *quantum nodes*. Both classical and quantum nodes can depend on tunable parameters  $\theta$  that we call *variables*, which are adjusted during optimization to minimize the cost. The nodes can receive inputs  $x$  from other nodes or directly from the global input to the hybrid computation, and they produce outputs  $f(x; \theta)$ . The computation can therefore be depicted as a Directed Acyclic Graph (DAG) that graphically represents the steps involved in computing the cost, which is produced by the final node in the DAG. By traversing the DAG, information about gradients can be accumulated via the rules of automatic differentiation. This is used to compute the gradient of the cost function with respect to all variables in order to minimize the cost with a gradient descent-type algorithm. It is important to note that automatic differentiation only requires a small constant overhead compared to the “forward” computation by collecting and reusing intermediate results. However, quantum nodes are black boxes to automatic differentiation, which means that accumulation of partial results does not extend to the interior of quantum nodes.

**Quantum nodes** While classical nodes (see Fig. 16(a)) can contain any numerical computations (of course, in order to differentiate the classical nodes, the computations have to be based on differentiable functions), quantum nodes have a more restricted layout.

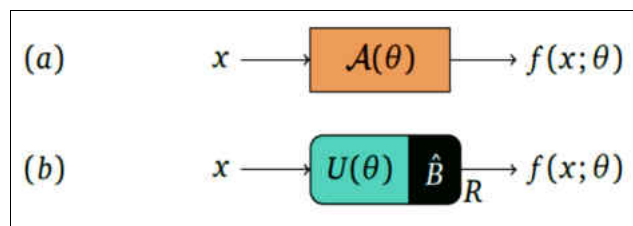


Figure 16. While a classical node consists of a numerical computation  $A$ , a quantum node executes a variational circuit  $U$  on a quantum device and returns an estimate of the expectation value of an observable  $\hat{B}$ , estimated by averaging  $R$  measurements.

A quantum node (in PennyLane represented by the QNode class) is an encapsulation of a function  $f(x; \theta): \mathbb{R}^m \rightarrow \mathbb{R}^n$  that is executed by means of quantum information processing on a *quantum device*. The device can either refer to quantum hardware or a classical simulator.

**Variational circuits.** The quantum device executes a parametrized quantum circuit called a *variational circuit* that consists of three basic operations:

- Prepare an initial state (here assumed to be the ground or vacuum state  $|0\rangle$ ).
- Apply a sequence of unitary gates  $U$  (or more generally, quantum channels) to  $|0\rangle$ . Each gate is either a fixed operation, or it can depend on some of the inputs  $x$  or the variables  $\theta$ . This prepares the final state  $U(x; \theta)|0\rangle$ .
- Measure  $m$  mutually commuting scalar observables  $\hat{B}_i$  in the final state.

Step 2 describes the way inputs  $x$  are encoded into the variational circuit, namely by associating them with gate parameters that are not used as trainable variables (This *input embedding* can also be interpreted as a feature map that maps  $x$  to the Hilbert space of the quantum system).

Step 3 describes the way quantum information is transformed back to the classical output of a quantum node as the expected values of the measured observables:  $f_i(x; \theta) = \langle \hat{B}_i \rangle = \langle 0 | U(x; \theta)^\dagger \hat{B}_i U(x; \theta) | 0 \rangle$ . The observables  $\hat{B}_i$  typically consist of a local observable for each wire (i.e., qubit or qumode) in the circuit, or just a subset of the wires. For example,  $\hat{B}_i$  could be the Pauli-Z operator for one or more qubits. The refined graphical representation of quantum nodes is shown in Fig. 16(b).

**Circuit architectures.** The heart of a variational circuit is the *architecture*, or the fixed gate sequence that is the skeleton of the algorithm. Three common types of architectures are sketched in Fig. 17.

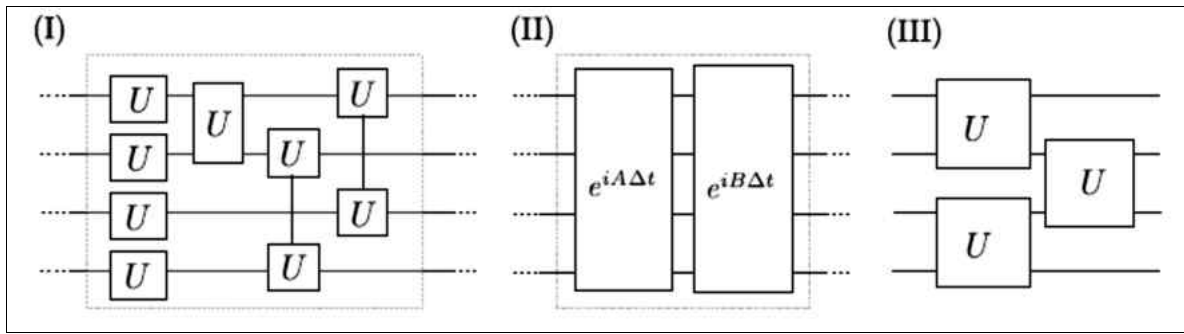


Figure 17. Different types of architectures for variational circuits: (I) layered gate architecture, (II) alternating operator architecture, and (III) an example of a tensor network architecture

The strength of an architecture varies depending on the desired use-case, and it is not always clear what makes a good ansatz. Investigations of the expressive power of different approaches are also ongoing. One goal of PennyLane is to facilitate such studies across various architectures and hardware platforms.

*Examples of hybrid optimization tasks.* Fig. 18 shows three examples of hybrid optimization tasks depicted as a DAG. Each of these models is available as a worked example in the PennyLane documentation.

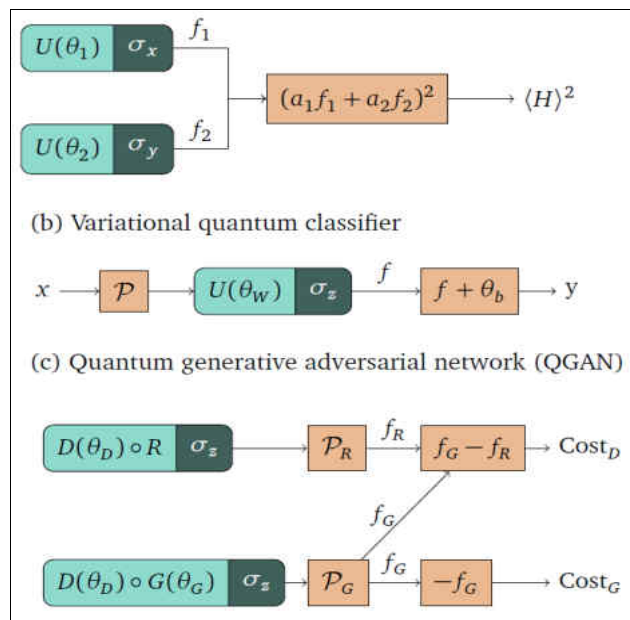


Figure 18. DAGs of hybrid optimization examples. These models and more are available as worked examples in the PennyLane docs.

Figure 18 (a) illustrates a variational quantum eigensolver, in which expectation values of two Pauli operators are combined with weights  $a_1, a_2$  to return the squared global energy expectation  $\langle H \rangle^2$ . Figure 18 (b) shows a variational quantum classifier predicting a label  $y$  given a data input  $x$  for a supervised learning task. The input is preprocessed by a routine  $\mathcal{P}$  and fed into a variational circuit with variables  $\theta_w$ . A classical node adds a bias variable  $\theta_b$  to the Pauli-Z expectation of a designated qubit. In Fig. 18 (c) one can see a quantum generative adversarial network (QGAN) example. It consists of two variational circuits. One represents the “real data” circuit  $R$  together with a discriminator circuit  $D$ , and the other has a “fake” generator circuit  $G$  replacing  $R$ . The result is postprocessed by  $\mathcal{P}_{R,G}$  and used to construct the cost function of the discriminator as well as the generator.

The goal of a GAN is to train the discriminator and generator in an adversarial fashion until the generator produces data that is indistinguishable from the true distribution.



**Computing gradients.** PennyLane focuses on optimization via gradient-based algorithms, such as gradient descent and its variations. To minimize the cost via gradient descent, in every step the individual variables  $\mu \in \theta$  are updated according to the following rule:

```

1: procedure GRADIENT DESCENT STEP
2:   for  $\mu \in \theta$  do
3:      $\mu^{(t+1)} = \mu^{(t)} - \eta^{(t)} \partial_{\mu} C(\theta)$ 

```

The learning rate  $\eta^{(t)}$  can be adapted in each step, depending either on the step number, or on the gradient itself.

**Backpropagating through the graph.** A step of gradient descent requires us to compute the gradient  $\nabla_{\theta} C(\theta)$  of the cost with respect to all variables  $\theta$ . The gradient consists of partial derivatives  $\partial_{\mu} C(\theta)$  with respect to the individual variables  $\mu \in \theta$ . In modern machine learning libraries like TensorFlow, PyTorch, or autograd, this computation is performed using automatic differentiation techniques such as the backpropagation algorithm. PennyLane extends these capabilities to computations involving quantum nodes, allowing computational models in these three machine learning libraries (including those with GPU-accelerated components) to seamlessly include quantum nodes. This makes PennyLane completely compatible with standard automatic differentiation techniques commonly used in machine learning.

While the backpropagation method — a classical algorithm — cannot resolve the quantum information inside quantum nodes, it is sufficient for us to compute the gradient or Jacobian of quantum nodes with respect to their (classical) inputs and variables. The key insight is to use the *same quantum device* (hardware or simulator) that implements a quantum node to also compute gradients or Jacobians of that quantum node.

Assume that only the node  $n^*$  depends on the subset of variables  $\theta \in \theta$ , and that  $\mu$  is in  $\theta$ . Let  $C \circ n_1^{(p)} \circ \dots \circ n^*$  be the path through the DAG of (quantum or classical) nodes that emerges from following the cost in the opposite direction of the directed edges until we reach node  $n^*$ . Since there may be  $N_p \geq 1$  of those paths (see Fig. 19), we use a superscript  $p$  to denote the path index.

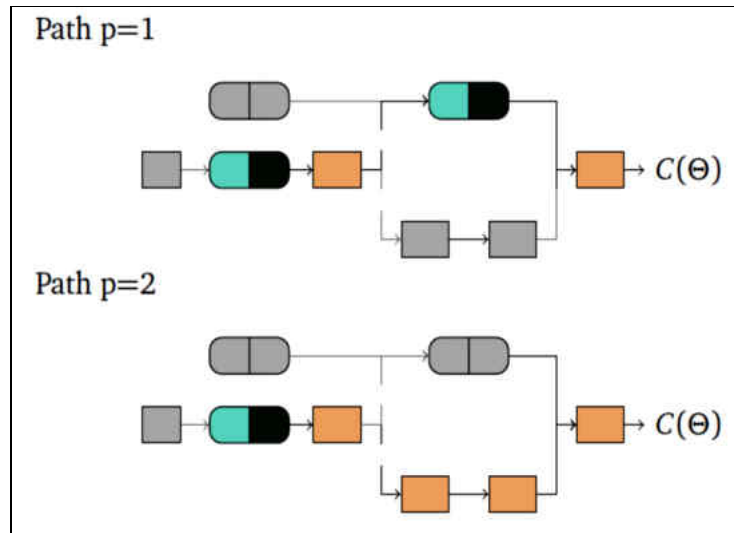


Figure 19. Example illustration of the two paths that lead from the cost function back to a quantum node

All branches that do not lead back to  $\theta$  are independent of  $\mu$  and can be thought of as constants.

The chain rule prescribes that the derivative with respect to the variable  $\mu \in \theta$  is given by

$$\partial_{\mu} C(\Theta) = \sum_{p=1}^{N_p} \frac{\partial C}{\partial n_1^{(p)}} \frac{\partial n_1^{(p)}}{\partial n_2^{(p)}} \dots \frac{\partial n^{*}}{\partial \mu}$$

While  $\partial_{\mu} C(\Theta)$  is a partial derivative and one entry of the gradient vector  $\nabla C(\Theta)$ , intermediate DAG nodes may map multiple inputs to multiple outputs. In this case, we deal with 2-dimensional Jacobian matrices rather than gradients.

In conclusion, we need to be able to compute two types of gradients for each node: the derivative  $\frac{\partial n_i^{(p)}}{\partial n_{i=1}^{(p)}}$  with respect to the input from a previous node, as well as the derivative with respect to a node variable  $\frac{\partial n}{\partial \mu}$ .

We actually refer to estimates of derivatives that result from estimates of expectation values. Numerically computed derivatives in turn are approximations of the true derivatives even if the quantum nodes were giving exact expectations (e.g., by using a classical simulator device).

**Derivatives of quantum nodes.** There are three types of methods to compute derivatives of quantum nodes with respect to a variable or input: analytical, numerical, or device-provided. By default, PennyLane uses the device or analytical derivatives wherever it can. Most types of quantum nodes support analytic derivatives, even if they are executed on quantum hardware.

**Analytic derivatives.** Recent advances in the quantum machine learning have suggested ways to estimate analytic derivatives by computing linear combinations of different quantum circuits. These rules are summarized and extended, which provides the theoretical foundation for derivative computations in PennyLane. In a nutshell, PennyLane makes two circuit evaluations, taking place at shifted parameters, in order to compute analytic derivatives. This recipe works for qubit gates of the form  $e^{-i\mu P}$ , where the Hermitian generator  $P$  has only two unique eigenvalues (which includes e.g., all single qubit rotation gates), as well as continuous-variable circuits with Gaussian operations. If  $f(x; \theta) = f(\mu)$  is the output of the quantum node, we have  $\partial_{\mu} f(\mu) = c(f(\mu + s) - f(\mu - s))$ , where  $c, s \in \mathbb{R}$  are fixed parameters for each type of gate. While this equation bears some structural resemblance to numerical formulas (discussed next), there are two key differences. First, the numbers  $c$  and  $s$  are not infinitesimal, but finite; second expression gives the *exact* derivatives. Thus, while analytic derivative evaluations are constrained by device noise and statistical imprecision in the averaging of measurements, they are not subject to numerical issues. To analytically compute derivatives of qubit gates or gates in a Gaussian circuit, PennyLane automatically looks up the appropriate derivative recipe (the numbers  $c$  and  $s$ ) for a gate, evaluates the original circuit twice (shifting the argument of the relevant gate by  $s$ ), subtracts the results, and scales by  $c$ .

**Numerical derivatives.** Numerical derivative methods require only ‘black-box’ evaluations of the model. We estimate the partial derivative of a node by evaluating its output,  $f(x; \theta) = f(\mu)$ , at several values which are close to the current value  $\mu \in \theta$  ( $\mu$  can be either a variable or an input here). The approximation of the derivative is given by  $\partial_{\mu} f(\mu) \approx \frac{f(\mu + \Delta\mu) - f(\mu)}{\Delta\mu}$  for the *forward finite-differences* method, and by

$$\partial_{\mu} f(\mu) \approx \frac{f\left(\mu + \frac{1}{2}\Delta\mu\right) - f\left(\mu - \frac{1}{2}\Delta\mu\right)}{\Delta\mu} \text{ for the } \textit{centered finite-differences} \text{ method.}$$

Of course, there is a tradeoff in choice of the difference  $\Delta\mu$  for noisy hardware.

**Device derivatives.** In addition to the analytic and numeric derivative implementations described above - which are supported by all simulator and hardware devices - PennyLane also supports directly querying the device for the derivative, if known. For example, a simulator written using a classical automatic differentiation library, such as TensorFlow or PyTorch, can make use of backpropagation algorithms internally to calculate derivatives. Compared to the analytic method on simulators, this may lead to significant time savings, as the information required to compute the derivative is stored and reused from the forward circuit evaluation—

simply adding constant overhead. Furthermore, the device derivative may also be used when interfacing with hardware devices that provide their own custom gradient formulations.

We consider a generic framework of optimization algorithms based on gradient descent. We consider a quantum algorithm that computes the gradient of a multi-variate real-valued function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  by evaluating it at only a logarithmic number of points in superposition. Our algorithm is an improved version of Jordan's gradient computation algorithm, providing an approximation of the gradient  $\nabla f$  with quadratically better dependence on the evaluation accuracy of  $f$ , for an important class of smooth functions. Furthermore, we show that most objective functions arising from quantum optimization procedures satisfy the necessary smoothness conditions, hence the algorithm provides a quadratic improvement in the complexity of computing their gradient. For a continuous phase-query model, the gradient computation algorithm has optimal query complexity up to poly-logarithmic factors, for a particular class of smooth functions. Moreover, for low-degree multivariate polynomials the algorithm can provide exponential speedups compared to Jordan's algorithm in terms of the dimension  $d$ .

Most quantum optimization procedures translate the objective function to the probability of some measurement outcome, and therefore evaluate it via sampling. To reflect this fact, we use an oracular model to represent our objective function, that is much weaker than the oracle model considered by Jordan. To be precise, we work with a coherent version of the classical random sampling procedure, i.e., we assume that the function is given by a *probability oracle*:

$$U_f : |x\rangle|0\rangle \rightarrow \sqrt{p(x)}|x\rangle|1\rangle + \sqrt{1-p(x)}|x\rangle|0\rangle, \text{ for every } x,$$

where the continuous input variable  $x$  is represented as a finite-precision binary encoding of  $x$ .

How many queries to  $U_p$  suffice to compute the *gradient* of  $p$ ?

Using empirical estimation it suffices to use  $O(1/\varepsilon^2)$  samples (obtained by querying  $U_p$ ) in order to evaluate  $p(x)$  with additive error  $\Theta(\varepsilon)$ . Provided that  $p$  is smooth we can compute an  $\varepsilon$ -approximation of

$\nabla_i p(x) = \frac{\partial p}{\partial x_i}$  by performing  $\tilde{O}(1)$  such function evaluations, using standard classical techniques. Hence, we

can compute an  $\varepsilon$ -approximation of the gradient  $\nabla p(x)$  with  $\tilde{O}(d)$  function evaluations of precision  $\Theta(\varepsilon)$ . The simple gradient descent algorithm uses  $TN$  gradient computations; therefore, the overall algorithm can be executed using  $\tilde{O}(TNd/\varepsilon^2)$  samples.

**Quantum speedups for the simple gradient descent algorithm.** Let us describe how to improve the query complexity of the simple gradient-descent algorithm, assuming that have access to a probability oracle of a smooth objective function  $p$ . Improve the complexity of  $\varepsilon$ -accurate function evaluations to  $O(1/\varepsilon)$  using amplitude estimation. Also improve the parallel search for finding a global minimum using the quantum minimum finding algorithm. Additionally, we present a quantum algorithm for gradient computation which quadratically improves the algorithm in terms of the dimension  $d$ . In particular, this shows that we can speed up the gradient-based optimization algorithm quadratically in almost all parameters, except the number of iterations  $T$ . The results are summarized below in Table 5.

Table 5. Quantum speedups for a simple gradient-descent algorithm

Method:	Simple algorithm	+Amp. est.	+Grover search	+This paper
Complexity:	$\tilde{O}(TNd/\varepsilon^2)$	$\tilde{O}(TNd/\varepsilon)$	$\tilde{O}(T\sqrt{N}d/\varepsilon)$	$\tilde{O}(T\sqrt{N}d/\varepsilon)$

**Improved quantum gradient computation algorithm** Stephen Jordan constructed a surprisingly simple quantum algorithm that can approximately calculate the  $d$ -dimensional gradient of a function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  with a *single* evaluation of  $f$ . In contrast, using standard classical techniques, one would use  $d + 1$  function evaluations to calculate the gradient at a point  $x \in \mathbb{R}^d$ : one can first evaluate  $f(x)$  and then, for every  $i \in [d]$

, evaluate  $f(x + \delta e_i)$  (for some  $\delta > 0$ ) to get an approximation of the gradient in direction  $i$  using the standard formula  $\nabla_i f(x) \approx \frac{f(x + \delta e_i) - f(x)}{\delta}$ .

Let us consider briefly Jordan algorithm.

**Overview of Jordan's algorithm.** The basic idea of Jordan's quantum algorithm is simple (see Part I, *Example: Jordan's numerical gradient algorithm*). First make two observations. Observe that if  $f$  is twice differentiable at  $x$ , then  $f(x + \delta) = f(x) + \nabla f \cdot \delta + O(\|\delta\|^2)$ , which in particular implies that for small  $\|\delta\|$ , the function  $f$  is very close to being affine linear. The second observation is that, using the value of  $f(x + \delta)$ , one can implement a phase oracle  $O_{2\pi Sf} : |\delta\rangle \rightarrow e^{2\pi i S f(x)} e^{2\pi i S \nabla f \cdot \delta} |\delta\rangle$  for a scaling factor  $S > 0$ , where the approximation uses  $f(x + \delta) \approx f(x) + \nabla f \cdot \delta$  for small  $\|\delta\|$ . The role of  $S$  is to make the phases appropriate for the final quantum Fourier transform.

**Sketch of the algorithm.** Assume that all real vectors are expressed up to some finite amount of precision. In order to compute the gradient at  $x$ , the algorithm starts with a uniform superposition

$$|\psi\rangle = \frac{1}{\sqrt{|G_x^d|}} \sum_{\delta \in G_x^d} |\delta\rangle$$

over the points of a sufficiently small discretized  $d$ -dimensional  $G_x^d$  around  $x$ , and applies the phase oracle  $O_{2\pi Sf}$  to  $|\psi\rangle$ .

Next, the inverse quantum Fourier transform is applied to the resulting state and each register is measured to obtain the gradient of  $f$  at  $x$  approximately. Due to approximate linearity of the phase, applying the Fourier transform will approximately give us the gradient. This algorithm uses  $O_{2\pi Sf}$  once and Jordan showed how to implement  $O_{2\pi Sf}$  using one sufficiently precise function evaluation. In order to improve the accuracy of the simple algorithm above, one could use some natural tricks. If  $f$  is twice continuously differentiable, it is easy to see that the smaller the  $G_x^d$  becomes, the closer the function gets to being linear. This gives control over the precision of the algorithm, however if we “zoom-in” to the function using a smaller grid, the difference between nearby function values becomes smaller, making it harder to distinguish them and thus increasing the complexity of the algorithm proportionally.

Also, it is well known that if the derivative is calculated based on the differences between the points  $(f(x - \delta/2), f(x + \delta/2))$  rather than  $(f(x), f(x + \delta))$ , then one gets a better approximation since the quadratic correction term cancels. To mimic this trick, Jordan chose a symmetric  $G_x^d$  around 0.

**Complexity of the algorithm.** For Jordan's algorithm, it remains to pick the parameters of the grid and the constant  $S$  in oracle  $O_{2\pi Sf}$ . For simplicity, assume that  $\|\nabla f(x)\|_\infty \leq 1$ , and suppose we want to approximate  $\nabla f(x)$  coordinate-wise up to  $\varepsilon$  accuracy, with high success probability. Under the *assumption* that “the 2<sup>nd</sup> partial derivatives of  $f$  have a magnitude of approximately  $D_2$ ”, Jordan argues that choosing  $G_x^d$  to be a  $d$ -dimensional hypercube with edge length  $\ell \approx \frac{\varepsilon}{D_2 \sqrt{d}}$  and  $N \approx \frac{1}{\varepsilon}$  with equally spaced grid points in each

dimension, the quantum algorithm yields an  $\varepsilon$ -approximate gradient by setting  $S = \frac{N}{\ell} \approx \frac{D_2 \sqrt{d}}{\varepsilon^2}$ . Moreover, since the Fourier transform is relatively insensitive to local phase errors it is sufficient to implement the phase  $Sf(x + \delta)$  up to some constant, say 1% accuracy.

During the derivation of the above parameters Jordan makes the assumption, that the third and higher-order terms of the Taylor expansion of  $f$  around  $x$  are negligible, however it is not clear from his work, how to actually handle the case when they are non-negligible. This could be a cause of concern for the runtime analysis, since these higher-order terms potentially introduce a dependence on the dimension  $d$ . Finally, in order to assess the complexity of his algorithm, Jordan considers the Binary oracle input model. This input model captures functions that are evaluated numerically using, say, an arithmetic circuit. Typically, the number of one and two-qubit gates needed to evaluate such functions up to  $n$  digits precision is polynomial in  $n$  and  $d$ . However, this input model does not fit the quantum optimization framework.

**Improvements of quantum algorithm.** Jordan argued that evaluating the function on a superposition of grid-points symmetrically arranged around 0 is analogous to using a simple central difference formula. Here also place the grid symmetrically, but realized that it is possible to directly use central difference formulas, which is the main idea behind our modified algorithm.

In applications of the gradient descent algorithm for optimization problems, it is natural to assume access to a phase oracle  $O_f : |x\rangle \rightarrow e^{if(x)} |x\rangle$  (allowing fractional queries) instead of the Binary access oracle  $B_f^n$ . If we wanted to use Jordan's original algorithm in order to obtain the gradient with accuracy  $\varepsilon$ , we need to implement the query oracle  $O_f^S$  by setting  $S \approx D_2 \sqrt{d} / \varepsilon^2$ , which can be achieved using  $\lceil S \rceil$  consecutive (fractional) queries. Although it gives a square-root dependence on  $d$  it scales as  $O(1/\varepsilon^2)$  with the precision. Employ the phase oracle model and improve the quadratic dependence on  $1/\varepsilon$  to essentially linear. Additionally, rigorously prove the square-root scaling with  $d$  under reasonable assumptions on the derivatives of  $f$ . For a class of smooth functions, the  $\tilde{O}(\sqrt{d} / \varepsilon)$ -query complexity is optimal up to poly-logarithmic factors.

**Analysis of Jordan's quantum gradient computation algorithm.** Describe Jordan's quantum gradient computation algorithm as [10-12].

---

**Algorithm 1** Jordan's quantum gradient computation algorithm

---

**Registers:** Use  $n$ -qubit input registers  $|x_1\rangle|x_2\rangle \cdots |x_d\rangle$  with each qubit set to  $|0\rangle$ .

**Labels:** Label the  $n$ -qubit states of each register with elements of  $G_n$  as in Definition 5.1.

**Input:** A function  $f : G_n^d \rightarrow \mathbb{R}$  with phase-oracle  $O_f$  access such that

$$O_f^{\pi^{2^{n+1}}} |x_1\rangle \cdots |x_d\rangle = e^{2\pi i 2^n f(x_1, x_2, \dots, x_d)} |x_1\rangle \cdots |x_d\rangle.$$

1: **Init** Apply a Hadamard transform to each qubit of the input registers.

2: **Oracle call** Apply the modified phase oracle  $O_f^{\pi^{2^{n+1}}}$  on the input registers.

3: **QFT**  $G_n^{-1}$  Fourier transform each register:

$$|x\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{k \in G_n} e^{-2\pi i 2^n xk} |k\rangle.$$

4: **Measure** each input register  $j$  and denote the measurement outcome by  $k_j$ .

5: **Output**  $(k_1, k_2, \dots, k_d)$  as the gradient estimate

---

**Algorithm 2** Jordan's quantum gradient computation algorithm**Registers:** Use  $n$ -qubit input registers  $|x_1\rangle|x_2\rangle\cdots|x_d\rangle$  with each qubit set to  $|0\rangle$ .**Labels:** Label the  $n$ -qubit states of each register with elements of  $G_n$  as in Definition 17.**Input:** A function  $f : G_n^d \rightarrow \mathbb{R}$  with phase-oracle  $O_f$  access such that

$$O_f^{2^{n+1}} |x_1\rangle|x_2\rangle\cdots|x_d\rangle = e^{2\pi i 2^n f(x_1, x_2, \dots, x_d)} |x_1\rangle|x_2\rangle\cdots|x_d\rangle.$$

- 1: **Init** Apply a Hadamard transform to each qubit of the input registers.
- 2: **Oracle call** Apply the modified phase oracle  $O_f^{2^{n+1}}$  on the input registers.
- 3:  $\text{QFT}_{G_n}^{-1}$  Fourier transform each register individually:

$$|x\rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{k \in G_n} e^{-2\pi i 2^n xk} |k\rangle.$$

- 4: **Measure** each input register  $j$  and denote the measurement outcome by  $k_j$ .
- 5: **Output**  $(k_1, k_2, \dots, k_d)$  as the estimation for the gradient.

If the relative precision of the approximation is precise enough, then Algorithm 2 can compute an approximation of  $g$  (the “gradient”) with small query and gate complexity (see, analysis of algorithms below). The gate complexity statement follows from the fact that the complexity of Algorithm 2 is dominated by that of the  $d$  independent quantum Fourier transforms, each of which can be approximately implemented using  $O(n \log n)$  gates. Repeat the procedure  $O(\log(d/\rho))$  times, which amounts to  $O(d \log(d/\rho) n \log n)$  gates. At the end we get  $d$  groups of numbers each containing  $O(\log(d/\rho))$  numbers with  $n$  bits of precision. Sort each group with a circuit having  $O(\log(d/\rho) \log \log(d/\rho) n \log n)$  gates. So, the final gate complexity is  $O(d \log(d/\rho) \log \log(d/\rho) n \log n)$ , which gives the stated gate complexity by observing that  $n = O(\log(M/\varepsilon))$ .

**Quantum singular value transformation.** Of the aforementioned quantum algorithms, quantum simulation is arguably the most diverse and rapidly developing. Within the last few years a host of techniques have been developed that have led to ever more powerful methods. The problem in quantum simulation fundamentally is to take an efficient description of a Hamiltonian  $H$ , an evolution time  $t$ , and an error tolerance  $\varepsilon$ , and find a quantum operation  $V$  such that  $\|e^{-iHt} - V\| \leq \varepsilon$  where the implementation of  $V$  should use as few resources as possible. The first methods introduced to solve this problem were Trotter formula decompositions and subsequently methods based on linear combinations of unitaries were developed to provide better asymptotic scaling of the cost of simulation. An alternative strategy was also developed concurrently with these methods that used ideas from quantum walks. Asymptotically, this approach is perhaps the favored method for simulating time-independent Hamiltonians because it is capable of achieving near-optimal scaling with all relevant parameters. The main tool developed for this approach is a walk operator that has eigenvalues  $e^{-i \arcsin(E_k/\alpha)}$ , where  $E_k$  is the  $k^{\text{th}}$  eigenvalue of  $H$  and  $\alpha$  is a normalizing parameter. While early work adjusted the spectrum to recover the desired eigenvalues  $e^{-iE_k}$  by using phase estimation to invert the arcsin, subsequent work achieved better scaling using linear combination of quantum walk steps. Recently another approach, called qubitization, was introduced to transform the spectrum in a more efficient manner based on a technique called quantum signal processing.

A new technique that was called as *quantum singular value transformation*, which unifies qubitization and quantum signal processing. The central object for this result is projected unitary encoding, which is defined as follows. Suppose that  $\tilde{\Pi}, \Pi$ , are orthogonal projectors and  $U$  is a unitary, then we say that the unitary  $U$  and the projectors  $\tilde{\Pi}, \Pi$  form a projected unitary encoding of  $A := \tilde{\Pi} U \Pi$ . The encoding is called symmetric if  $\tilde{\Pi} = \Pi$ .



Roughly speaking qubitization turns a symmetric projected unitary encoding  $\Pi U \Pi = H$  of a Hermitian operator  $H$  into a unitary  $V$  which is the square root of a (Szegedy-type) quantum walk operator  $U(2\Pi - I)U^\dagger(2\Pi - I)$ , so that each eigenvector  $|\psi\rangle$  of  $H$  with eigenvalue  $\lambda$  becomes a superposition of two eigenvectors  $|\psi^\pm\rangle$  of  $V$  with eigenvalues  $e^{\pm i \arccos \lambda}$ . If  $U$  happens to be a Hermitian unitary (i.e., a reflection operator), then one can use  $V := U(2\Pi - I)$ ; otherwise one can replace  $U$  by a suitable Hermitian unitary  $\tilde{U}$  using a controlled- $U$  and controlled- $U^\dagger$  gate.

Finally, using quantum signal processing one can transform the spectrum of the unitary  $V$ , resulting in a new circuit  $U'$ , which is a projected unitary encoding of a transformed operator  $P(H)$  (see Fig. 20).

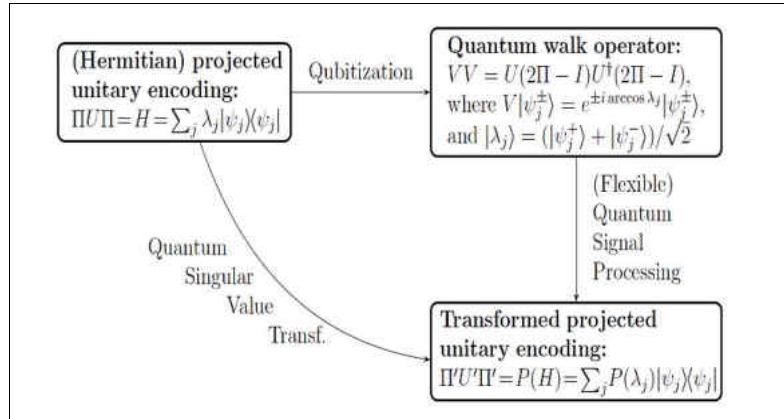


Figure 20. Schematic comparison of QSVT with previous approaches

Generalize and unify the above two steps in quantum singular value transformation is possible by directly implementing the transformation (see, Figs 21 and 22) on the singular values of  $H$ . The result works for arbitrary matrices and does not require symmetric encodings, (i.e.,  $\tilde{\Pi} \neq \Pi$  is allowed).

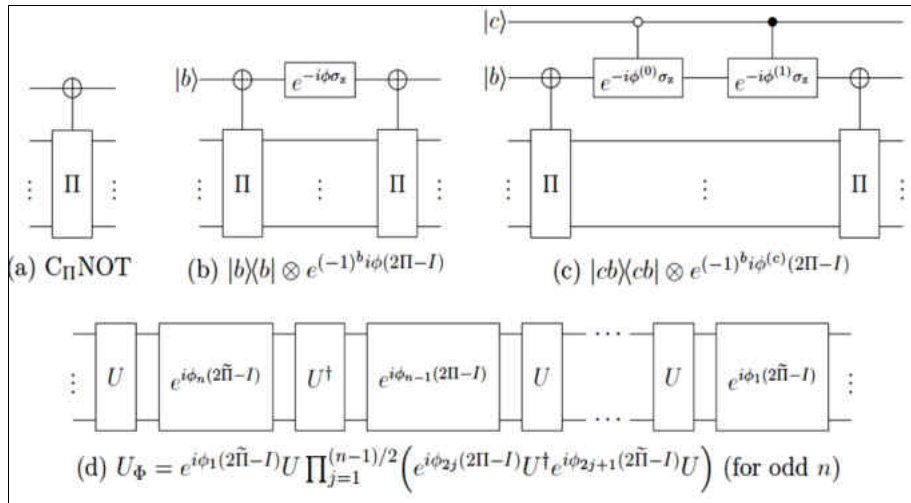


Figure 21. Gates and gate sequences used for singular value transformation

Figure 21 (a) shows how to implement a  $C_{\Pi} \text{NOT}$  gate, and Fig. 21 (b) shows how to implement  $e^{i\phi(2\Pi-I)}$  using a single ancilla qubit, two  $C_{\Pi} \text{NOT}$  gates and an  $e^{-i\phi\sigma_z}$  gate. Figure 21 (c) demonstrates how to implement a controlled version of the gate  $e^{i\phi^{(c)}(2\Pi-I)}$ , by only controlling the single-qubit gate  $e^{-i\phi^{(c)}\sigma_z}$ . Finally, Fig. 21 (d) summarizes the complete circuit. What remains is to discuss how to efficiently implement alternating phase modulation sequences. The operator  $e^{i\phi(2\Pi-I)} = C_{\Pi} \text{NOT} (I \otimes e^{i\phi\sigma_z}) C_{\Pi} \text{NOT}$  can be implemented using a

single ancilla qubit, two uses of  $C_{\Pi}$ NOT, and a single-qubit phase gate  $e^{i\phi\sigma_z}$ , leading to an efficient implementation of  $U_{\Phi}$ , see Fig. 21 (b).

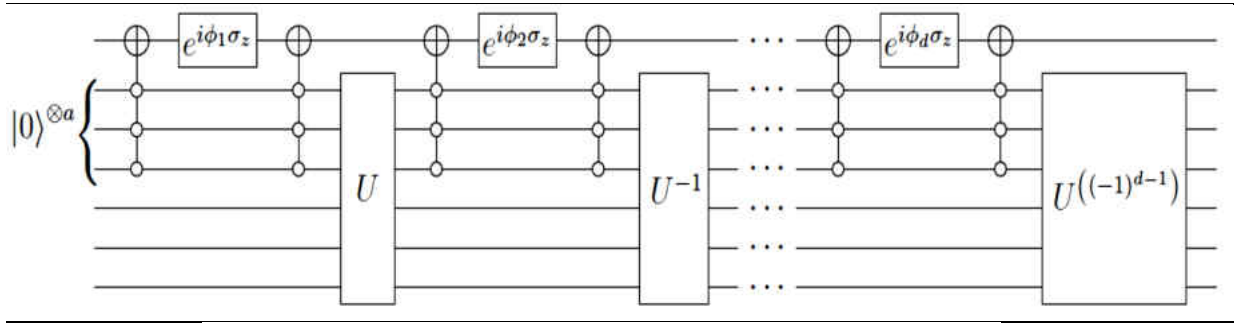


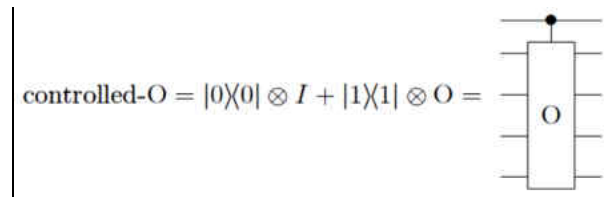
Figure 22. Circuits used for quantum singular value transformation

Moreover, do not need  $U$  to be Hermitian, and in case  $P$  is an even or odd polynomial, do not even require access to controlled versions of  $U$  or  $U^{\dagger}$ . The empty dots denote control by the  $|0\rangle$  state, so that the corresponding gate is an “inverted” Toffoli gate, where each qubit is conjugated by an  $X$  gate compared to the usual Toffoli gate. The other gates are single-qubit rotations or applications of  $U$  or  $U^{\dagger}$ . The structure is very similar to the amplitude amplification circuit of Fig. 22.

Other algorithms can also be cast in the singular value transformation framework, including optimal Hamiltonian simulation, robust oblivious amplitude amplification, fast QMA amplification, fast quantum OR lemma and certain quantum walk results. Based on these techniques we also show how to exponentially improve the complexity of implementing fractional queries to unitaries with a gapped spectrum.

### Additional necessary definitions from quantum algorithm complexity

**Quantum queries.** It is often useful to think about quantum algorithms in a modular fashion, and study how efficiently a problem can be solved using a particular subroutine. Since we want to separate the implementation issues of the subroutine from the higher-level algorithms, we often assume that we have access to a quantum subroutine in the form of a *quantum oracle*  $\mathbf{O}$ , which is essentially a black-box unitary circuit. The only way the quantum circuit is allowed to extract information from such an oracle  $\mathbf{O}$  is by using it on some of the circuit’s qubits. We usually assume that both the oracle and its inverse can be applied in a controlled fashion, i.e., the circuit can use  $|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes \mathbf{O}$  and  $|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes \mathbf{O}^{\dagger}$  gates as well (For a linear operator  $A$  we use  $A^{\dagger}$  to denote its adjoint;  $A$  is unitary if and only if  $A^{\dagger} = A^{-1}$ ). If  $\mathbf{O}$  is, say, a 4-qubit unitary, then the corresponding controlled quantum gate is denoted by



The query complexity of a particular quantum circuit that solves a problem using an oracle  $\mathbf{O}$  is the number of uses of controlled- $\mathbf{O}$  and controlled- $\mathbf{O}^{\dagger}$  gates in the circuit, whereas the gate complexity is the number of additional single and two-qubit gates used. The *quantum query complexity* of some problem is the minimum query complexity over all quantum circuits that solve the problem (whatever that means in the given context).

**Remark.** In some cases, also assumed access to *quantum read-only memory* (QROM), which is essentially an oracle that gives cheap access to a vector  $x \in \{0,1\}^N$  and for all  $i \in \{0,1,\dots,N-1\}$  provides access to  $x_i$  by mapping  $|i\rangle|b\rangle \mapsto |i\rangle|b \oplus x_i\rangle$ . In some cases when explicitly say it, QROM queries just count as simple quantum gates and incorporate them as part of the gate complexity. However, in general building a QROM requires roughly  $N$  single- and two-qubit gates.

**Grover search and amplitude amplification.** In the black-box search problem, we are promised that there is a single good element among  $N$  elements, and we have a black box which can tell for each element whether it is good or not. It is not hard to show that classically one needs to make at least around  $N$  queries to the black box in order to find the marked element with high probability. However, if we can make quantum queries to the black-box, then roughly  $\sqrt{N}$  queries suffice, as shown by Grover's search algorithm. This is essentially optimal, as shown by the hybrid-method argument of Bennett. Thus, the quantum query complexity of black-box search is about  $\sqrt{N}$ .

Grover's search algorithm follows from a more general technique called *amplitude amplification*. Suppose that we have a black-box unitary  $U: |0\rangle^{\otimes k} \mapsto |\psi\rangle = \sqrt{p}|1\rangle|\psi_{\text{good}}\rangle + \sqrt{1-p}|0\rangle|\psi_{\text{bad}}\rangle$  that prepares some state  $|\psi_{\text{good}}\rangle$  with success probability  $p$ , and marks success with a  $|1\rangle$  flag qubit. For example, in the search problem one can first prepare a uniform superposition  $\frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle$  over all elements, and then make a query. If the marked element is  $m \in \{0, 1, \dots, N-1\}$ , then the resulting state is  $\frac{1}{\sqrt{N}}|1\rangle|m\rangle + \frac{\sqrt{N-1}}{\sqrt{N}}|0\rangle\left(\frac{1}{\sqrt{N-1}} \sum_{j \neq m} |j\rangle\right)$ , so that  $|\psi_{\text{good}}\rangle = m$  is the basis state for the unique marked element. The classical approach for preparing the state with high probability would be to use the procedure roughly  $1/p$  times, and stop once the flag qubit is measured in the  $|1\rangle$  state. However, the problem can be solved with only roughly  $\sqrt{1/p}$  uses of  $U$  and  $U^\dagger$  with the help of the Grover operator  $G_U := U(2|0\rangle\langle 0|^{\otimes k} - I_k)U^\dagger((2|0\rangle\langle 0| - I_1) \otimes I_{k-1})$  (see Fig. 23) where  $I_\ell$  denotes the identity operator on  $\ell$  qubits.

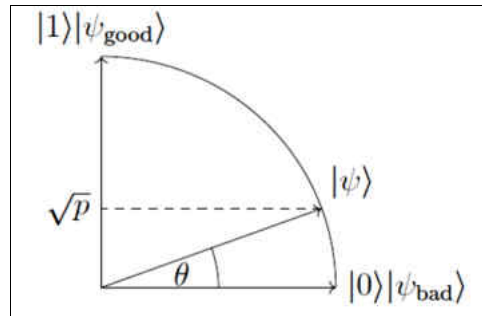


Figure 23. Geometric illustration of the Grover operator  $G_U$

Let,  $\theta := \arcsin(\sqrt{p})$ , then we get that  $G_U$  acts as a rotation by angle  $2\theta$  on the subspace spanned by  $|1\rangle|\psi_{\text{good}}\rangle$ , and  $|0\rangle|\psi_{\text{bad}}\rangle$ . Indeed, within this subspace  $((2|0\rangle\langle 0| - I_1) \otimes I_{k-1})$  is a reflection about  $|0\rangle|\psi_{\text{bad}}\rangle$ , and  $U(2|0\rangle\langle 0|^{\otimes k} - I_k)U^\dagger$  is a reflection about  $|\psi\rangle$ .

The product of these reflections is a rotation by angle  $2\theta$ , which can be seen for example considering the image of  $|0\rangle|\psi_{\text{bad}}\rangle$  and  $|\psi\rangle$ , see Fig. 24.

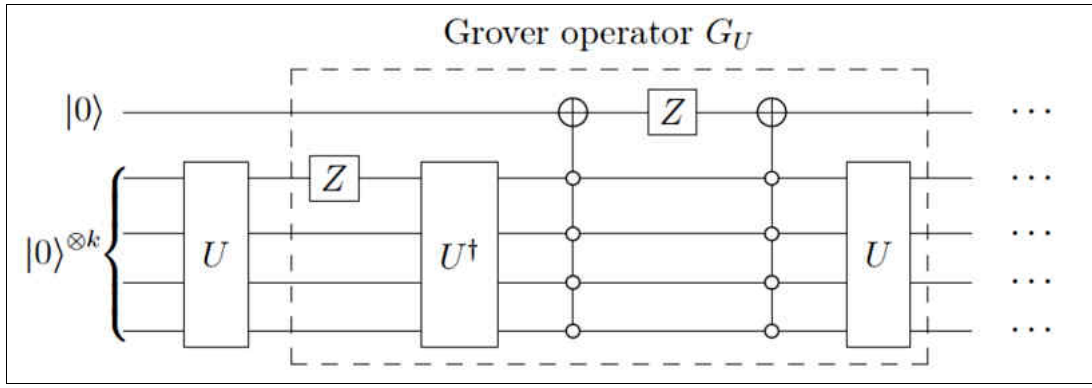


Figure 24. Amplitude amplification using the Grover operator

Thus,  $G_U^{\frac{\pi}{4\theta} - \frac{1}{2}} U |0\rangle^{\otimes k} = |1\rangle |\psi_{\text{good}}\rangle$ . So, applying  $G_U$  roughly  $\frac{\pi}{4\theta} - \frac{1}{2}$  times solves the problem with high

probability. However, if  $p$  is only a lower bound on the success probability then this operator might apply too much rotation, and the success probability might be small. This issue can be overcome by applying  $G_U$  a random number of times, and repeating the procedure a few times to boost the success probability. A more direct solution to this problem is to use the so-called fixed-point amplitude amplification algorithm, which provides a single-shot solution without using multiple repetitions. In quantum circuit diagrams the “time flows from left to right”, i.e., in the above circuit we start with the quantum state  $|0\rangle^{\otimes k}$  and append a single ancilla qubit, then apply  $U$ . The following gates implement the Grover operator  $G_U$  with the help of the ancilla qubit. The empty dots denote control on the  $|0\rangle$  state and so the  $(k+1)$ -qubit gates surrounding the second  $Z$  gate are reversible OR gates, but we can also think about them as (inverted) generalized Toffoli gates.

**Quantum Fourier transform.** The Quantum Fourier Transform (QFT) on  $N$  elements is the quantum analogue of the Discrete Fourier Transform (DFT) on  $\mathbb{Z}_N$ . If  $N = 2^n$ , then QFT becomes an  $n$ -qubit unitary

defined as follows:  $QFT_n : |j\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{k \in \{0,1,\dots,2^n-1\}} e^{2\pi i \frac{jk}{N}} |k\rangle$ . This unitary can be built using about  $n^2$  single-

and two-qubit gates, which number can be reduced to about  $n \log(n)$  if one only needs a  $(1/\text{poly}(n))$ -approximation. (In contrast the classical Fast Fourier Transform (FFT) algorithm has complexity about  $2^n n$ , which is exponentially larger.) The fact that QFT has such an efficient implementation provides the basis of many important quantum algorithms including Shor’s famous integer factoring algorithm.

## A generic model of quantum optimization algorithms

Variational quantum algorithms designed for quantum optimization and machine learning procedures have the following core idea: they approximate an optimal solution to a problem by tuning some parameters in a quantum circuit. The circuit usually consists of several simple gates, some of which have tunable real parameters, e.g., the angle of single qubit (controlled) rotation gates. Often, if there are enough tunable gates arranged in a nice topology, then there exist parameters that induce a unitary capable of achieving a close to optimal solution. In such variational approaches, one can decompose the circuit into three parts each having a different role (see Fig. 25). The circuit starts with a state preparation part which prepares the initial quantum state relevant for the problem. We call this part ‘Prep.’ in Fig. 25. The middle part consists of tunable parameters  $x$  and fixed gates, which are together referred to as ‘Tuned’ in Fig. 25.

The circuit on the top left has classically set parameters  $|x\rangle$  (represented as a vector of fixed point binary numbers), whereas the circuit on the top right has parameters  $x$  described by an array of qubits  $|x\rangle$ . The black squares connected to the ‘Tuned’ circuit indicate non-trivial control structure for which an example is presented on the bottom figure, showing how to implement a quantumly tunable rotation gate built from simple controlled rotation gates. Finally, there is a verification circuit that evaluates the output state, and marks success if the

auxiliary qubit is  $|1\rangle$ . We denote the verification process by  $V$  in Fig. 25. The quality of the circuit (for parameter  $x$ ) is assessed by the probability of measuring the auxiliary qubit and obtaining 1. One can think of the tunable circuit as being tuned in a classical way as shown in Fig. 25 (a) or a quantum way as in Fig. 25 (b). In the classical case, the parameters can be thought of as being manually set. Alternatively, the parameters can be quantum variables represented by qubits. The advantage of the latter is that it allows us to use quantum techniques to speedup optimization algorithms. However, the drawback is that it requires more qubits to represent the parameters and requires implementation of additional controlled-gates, see for example, Fig. 25 (c).

Let us denote by  $U(x)$  the circuit in Fig. 25 (a) and the corresponding circuit in Fig. 25 (b) as  $U := \sum_x |x\rangle\langle x| \otimes U(x)$ . The goal in these optimization problems is to find the optimal parameters (i.e.,  $x$ ) which maximize the probability of obtaining 1 after the final measurement, thereby solving the problem  $\arg \max_x p(x)$ , where  $p(x) = \left\| (|1\rangle\langle 1| \otimes I) U(x) |\vec{0}\rangle \right\|^2$ .

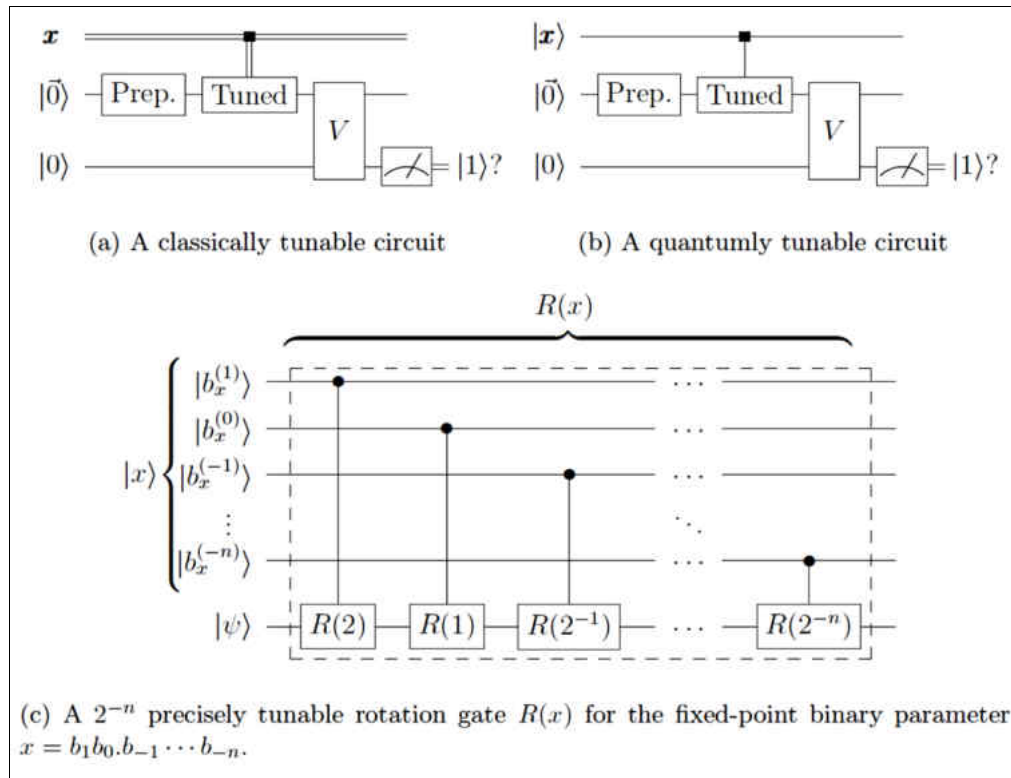


Figure 25. Two different approaches to tunable quantum optimization

A well-known technique to solve continuous-variable optimization problems like the one above is gradient-ascent. In practice, gradient-based methods represent one of the most commonly used paradigms for solving continuous optimization problems.

As we discussed earlier, finding globally optimal parameters for optimization problems is often hard. Therefore, in practice one usually relies on heuristics to find an approximate solution  $x_a$  such that  $p(x_a)$  is close to optimal. There are several heuristic optimization techniques that are often applied to handle such problems. One of the most common techniques is gradient ascent, which follows a greedy strategy to obtain the optimal solution. It simply follows the path of steepest ascent on the landscape of the objective function to find a solution that is, at least up to small local perturbations, optimal.

Such solutions are called locally optimal. A quadratic speedup in  $d$  – the number of control parameters. For this we also need to use a quantumly controlled circuit, but the overhead in the number of qubits is much smaller than in the previous Grover-type speedup. The underlying quantum technique crucially relies on the quantum Fourier transform as it is based on an improved version of Jordan's gradient computation algorithm. It can be optionally combine this speedup with the above-mentioned maximum finding improvement, which



then gives a quantum algorithm that uses the quantumly controlled circuit (Fig. 25 (b))  $\tilde{O}(T\sqrt{Nd}/\varepsilon)$  times, and achieves essentially the same guarantees as the classical algorithm.

Therefore, we can achieve a quadratic speedup in terms of all parameters except in  $T$  and obtain an overall complexity of  $\tilde{O}(T\sqrt{Nd}/\varepsilon)$ . For a summary of the speed ups see Table 5.

The most basic improvement which works even for classically controlled circuits (Fig. 25 (a)) is to estimate the probability  $p(x)$  in Step 5 using quantum amplitude estimation rather than doing repeated measurements and taking the average. If one wants to determine the value  $p(x)$  up to error  $\varepsilon$  for some fixed  $x$ , the quantum approach uses the circuit  $O(1/\varepsilon)$  times, whereas the classical statistical method would require  $\Omega(1/\varepsilon^2)$  repetitions, due to the additive property of variances of uncorrelated random variables. Although this is a natural improvement, which does not require much additional quantum resources, many papers that describe a similar procedure do not mention it. There is a drawback using Grover search-based techniques. The disadvantage of quantum maximum finding approach over classical methods is, that the amount of time it takes to reach a local maximum using the gradient ascent might vary a lot. The reason is that classically, once we reached a local maximum, we can start examining the next starting point, whereas if we use Grover search we do the gradient updates in superposition so we need to run the procedure for the largest possible number of gradient steps. To reduce this disadvantage, one could use variable time amplitude amplification techniques.

### ***Quantum Circuit Parameters Learning with Gradient Descent Using Backpropagation***

Quantum-classical hybrid algorithm needs to build an efficient simulation channel to connect ‘the quantum and the classic’ organically. In QAOA, VQE, or other hybrid NISQ algorithms, there exists a challenging task to optimize the model parameter. In all these algorithms, the parameter search and updating are performed in the classical computer. In a complete classical approach, the optimal parameter search is usually categorized as a mathematical optimization problem, where various methods both gradient-based and non-gradient based have been widely utilized. For the quantum circuit learning, so far, most of parameter searching algorithm is based on non-gradient. However, recently, gradient-based ones such as SPSA and a finite difference method have been reported. An error backpropagation algorithm on quantum circuit learning to calculate the gradient required in parameter optimization efficiently developed. The purpose of this work is to develop a gradient-based circuit learning algorithm with superior learning speed to the ones reported so far. The error backpropagation method is known as an efficient method for calculating gradients in the field of deep neural network machine learning when updating parameters using the gradient descent method. Further speed improvement can be easily realized through using the GPGPU technique, which is again well established and under significant growth in the field of deep learning. The idea behind the proposal is described as follows. By carefully examining the simulation process of a quantum circuit, if the input quantum state is  $|\psi_{in}\rangle$  and a specific quantum gate  $U(\theta)$  is applied as shown in Fig. 26, the output state  $|\psi_{out}\rangle$  can be expressed by the dot product of the input state and quantum gate  $|\psi_{out}\rangle = U(\theta)|\psi_{in}\rangle$ .

On the other hand, the calculation process of a fully connected neural network without activation function can be written as  $\mathbf{Y} = \mathbf{W} \cdot \mathbf{X}$ , where  $\mathbf{X}$  is the input vector,  $\mathbf{W}$  is the weight matrix of the network, and  $\mathbf{Y}$  is the output. It can be seen that the quantum gate  $(\theta)$  is very similar to the network weight matrix  $\mathbf{W}$ . This shows that backpropagation algorithms that are used for deep neural networks can be modified to some extent to be applied to the simulation process of quantum circuit learning.

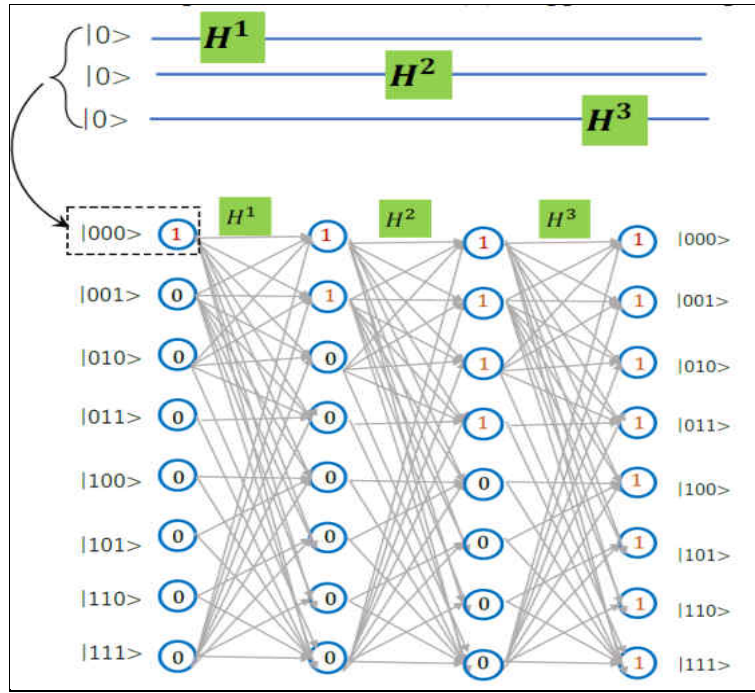


Figure 26. Example of three gates quantum circuit and its corresponding fully connected quantum network, showing similarity to a four-layer-neural network with an equal number of nodes in the input layer, middle layer and output layer. Note that the amplitude value is not normalized for better eye-guiding illustration

The method makes it possible to reduce the time significantly for gradient calculation when the number of qubits is increased, or the depth of the circuit (the number of gates) is increased. Meanwhile, by taking advantage of GPGPU, which is developed upon backpropagation, it is expected that using gradient-based backpropagation in the NISQ hybrid algorithm will further facilitate parameter search when many qubits and deeper circuits are deployed.

**Quantum Backpropagation Algorithm.** As shown in Fig. 26, a quantum circuit can be effectively represented by a fully connected quantum network with significant similarity to the conventional neural network except for the two facts: 1) there is no activation function applied upon each node, so the node is not considered as a neuron (or assuming an identical activation function); 2) the number of nodes are equal among all the input layer, middle layer as well as output layer, so there is no notion of ‘hidden’ layer since the dimensionality of each layer is the same. Noticed that the states shown as input in the quantum circuit is only one of the  $2^n$  ( $n$  is the number of qubits) with the amplitude of ‘1’ (not normalized), see Fig. 26 for details. The network similarity implies that the learning algorithm, such as the backpropagation heavily used in the field of deep machine learning, can be shared by the quantum circuit as well.

In general, the backpropagation method uses the chain rule of a partial differential to propagate the gradient back from the network output and calculate the gradient of the weights. Owing to the chain rule, the backpropagation can be done only at the input/output relationship at the computation cost of a node. In the simulation of quantum computing by error backpropagation, the quantum state  $|\psi\rangle$  and the quantum gates are represented by complex value. The derivation details regarding the quantum backpropagation in complex-valued vector space. When the input of  $n$  qubits is  $|\psi_{in}\rangle$  and the quantum circuit parameter network  $W(\theta)$  is

applied, the output  $|\psi_{out}\rangle$  can be expressed as:  $W(\theta)|\psi_{in}\rangle = \sum_{j=0}^{2^n-1} c_{\theta}^j |j\rangle = |\psi_{out}\rangle$ , where  $c_{\theta}^j$  is the probability

amplitude of state  $|j\rangle$  and  $|c_{\theta}^j|^2 = p_{\theta}^j$  is the observation probability of state  $|j\rangle$ . If loss function  $L$  can be expressed by using observation probability determined by quantum measurement, the gradient of the learning parameter can be described as:  $\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial p_{\theta}^j} \frac{\partial p_{\theta}^j}{\partial \theta}$ . Since  $p_{\theta}^j = |c_{\theta}^j|^2 = c_{\theta}^j \bar{c}_{\theta}^j$ , where  $\bar{c}_{\theta}^j$  is the conjugate of  $c_{\theta}^j$ .

Therefore, the gradient of observation probability can be further expanded as:

$$\frac{\partial p_{\theta}^j}{\partial \theta} = \frac{\partial c_{\theta}^j \bar{c}_{\theta}^j}{\partial \theta} = \bar{c}_{\theta}^j \frac{\partial c_{\theta}^j}{\partial \theta} + c_{\theta}^j \frac{\partial \bar{c}_{\theta}^j}{\partial \theta} \text{ or can be further expanded as: } \bar{c}_{\theta}^j \frac{\partial c_{\theta}^j}{\partial \theta} + c_{\theta}^j \frac{\partial \bar{c}_{\theta}^j}{\partial \theta} = \bar{c}_{\theta}^j \frac{\partial c_{\theta}^j}{\partial \theta} + \bar{c}_{\theta}^j \frac{\partial c_{\theta}^j}{\partial \theta}$$

Therefore,  $\frac{\partial L}{\partial \theta} = 2 \operatorname{Re} \left[ \frac{\partial L}{\partial p_{\theta}^j} \frac{\partial p_{\theta}^j}{\partial c_{\theta}^j} \frac{\partial c_{\theta}^j}{\partial \theta} \right]$  and  $\left[ \frac{\partial L}{\partial p_{\theta}^j} \frac{\partial p_{\theta}^j}{\partial c_{\theta}^j} \frac{\partial c_{\theta}^j}{\partial \theta} \right]$  can be obtained by error backpropagation in

the same way as the conventional calculation used in deep neural network. Meanwhile, one advantage of the proposed method is that the quantum gate matrix containing complex value is converted to real value. The gradient of the loss function with respect to  $\theta$  can be obtained from the real part of the value of the complex vector space calculated by the conventional backpropagation.

Implementing architecture when using a real machine such as NISQ type quantum described in Fig. 27. To use the error backpropagation method, a quantum state  $|\psi\rangle$  is required to get prepared at the initial stage. Therefore, as shown in the Fig. 27, a quantum circuit having the same configuration as the real quantum circuit must be prepared as a quantum simulator on a classical computer. It should be noticed here that this could not be considered as additional load for the quantum computing scientist since a quantum computer is not allowed to be disturbed during the working condition, unlike the classical computer, it needs its counterpart of quantum circuit simulator to monitor and diagnose the qubits and gate error and characterizing the advantage of quantum computers over classical computers. Therefore, for a real quantum computer, it always requires a quantum simulator ready for use at any time.

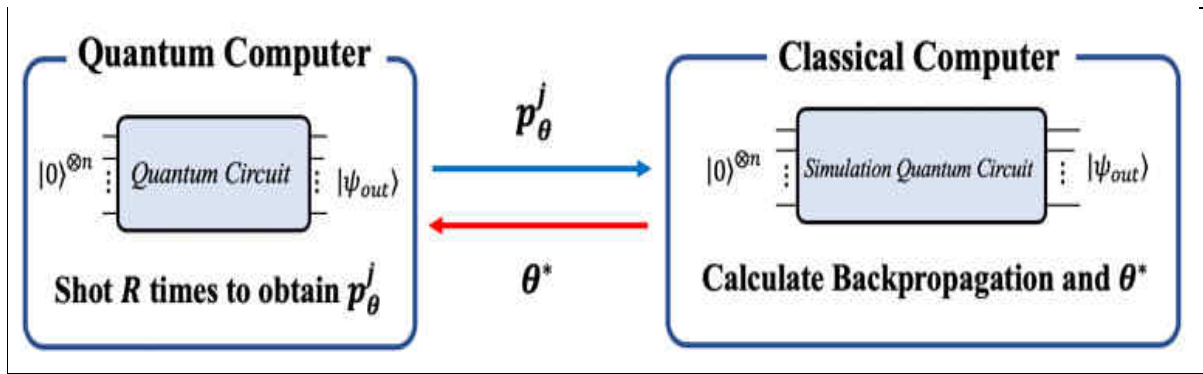


Figure 27. Implementation architecture of error backpropagation-based quantum circuit learning on the real NISQ type quantum computer [13]

That means we can always access the quantum simulator, as shown on the right side of Fig. 27, to examine and obtain detailed information regarding the performance of the corresponding real quantum computer. Observation probability for each state can be calculated by shooting times at the real quantum computer side. The observation probability obtained from the real quantum machine is then passed to the classical computer, and the quantum circuit in the simulator for simulation is then used. The parameter  $\theta$  can be updated using backpropagation since all the intermediate information is available at the simulator side. After the parameter  $\theta^*$  is updated at the simulate side, it will return to the real quantum machine for next iteration quantum simulation.

### Small quantum computers and large classical data sets

Hybrid classical-quantum algorithms for problems involving a large classical data set  $X$  and a space of models  $Y$  such that a quantum computer has superposition access to  $Y$  but not  $X$ . These algorithms use data reduction techniques to construct a weighted subset of  $X$  called a coreset that yields approximately the same loss for each model. The coreset can be constructed by the classical computer alone, or via an interactive protocol in which the outputs of the quantum computer are used to help decide which elements of  $X$  to use. By using the quantum computer to perform Grover search or rejection sampling, this yields quantum speedups for maximum likelihood estimation, Bayesian inference and saddle-point optimization. Concrete applications include  $k$ -means clustering, logistical regression, zero-sum games and boosting. Existing quantum algorithms

for optimization and machine learning are often less complete than their classical counterparts because they do not use realistic models of their input data.

One way to view this is that they are meant to be subroutines in larger “end-to-end” algorithms that will provide the data in the needed format. However, there has been relatively little research on these more complete algorithms and their development has often been nontrivial. Indeed, the trivial methods of turning large classical datasets into either quantum oracles or quantum states are so expensive as to negate any possible quantum advantage. As a result, even Grover's algorithm has not yet been successfully applied to speed up any natural machine learning task, despite being arguably the simplest, most widely known, and most widely applicable quantum algorithm.

One proposal is to use a “quantum RAM,” typically meaning a large classical memory which can be queried in superposition. This enables powerful quantum algorithms to be used, often with provable speedups. However, current classical computing or data storage hardware does not function as quantum RAM, and near-term hardware plans by leading groups using trapped ions, superconductors, or photons on chips do not involve quantum RAM. Also, building a large quantum RAM may run into many of the same challenges that occur in building a large universal quantum computer. At first glance it seems that quantum advantage requires problems with small input sizes. If the input size is  $n$  then it is hard to avoid expending effort proportional to  $n$  even before the computation starts, in order to acquire, store and load the input. Even alternate models such as property testing or streaming can typically mitigate only some of these. Similar issues apply to large outputs. For problems that can be solved on classical computers in time  $\tilde{O}(n)$ , it would seem that there is little scope for quantum advantage. For this reason, proposals for quantum advantage usually involve problems where the best-known classical runtime scales rapidly with the input size, perhaps exponentially.

How can quantum computers be of use in this setting and What can a quantum computer do with a large classical data set?

It will explore the ability of small quantum computers to work together with large classical computers to analyze large data sets. Work in a regime where the input size is  $n$ , the classical computer runs in time nearly linear in  $n$ , and do not use any unconventional access models for the quantum computer. Since qubits are likely to always remain more expensive than bits, this hybrid classical-quantum model should be relevant even when quantum computers with millions or billions of qubits are available.

At the same time, many tasks in classical computing, such as machine learning, are evolving towards the use of increasingly large data sets, for which a runtime of even  $O(n^2)$  can be infeasible.

The key principle will be *data reduction*. We will approximate a data set  $X = \{x_1, \dots, x_n\}$  with a much smaller subset  $X'$  (called a “coreset”) along with a weight function  $w: X' \rightarrow \mathbb{R}_{\geq 0}$  such that  $X', w$  can adequately substitute for  $X$  in solving problems of interest. This can be achieved either by a classical computer alone, or a classical computer together with a quantum computer. Then a hard optimization problem can be solved on the quantum computer using the reduced data set  $X'$  [1]. Data reduction can be useful for any quantum optimization algorithm for which computing the objective function requires examining a large data set.

*Example: Minimizing empirical loss with Grover.* Suppose a set of data points  $X$ , a set of models  $Y$ , and a loss function  $f: X \times Y \mapsto \mathbb{R}$  are given. The goal is to compute  $\arg \min_{y \in Y} \sum_{x \in X} f(x, y)$ , i.e., to choose the model which minimizes the empirical loss. It is important to emphasize the form of the input:  $X$  is an explicit data set  $x_1, \dots, x_n$ ;  $Y$  is a set that may be large or infinite but has a succinct description, e.g.  $Y$  might be the set of all ways of choosing a mixture of up to  $k$  Gaussians in  $\mathbb{R}^d$ ;  $f$  is given as an explicit and short algorithm. To apply Grover's algorithm (technically the Durr-Hoyer algorithm for minimizing a black-box function) will require  $O(\sqrt{|Y|})$  evaluations of  $F(y) := \sum_{x \in X} f(x, y)$ . However, each evaluation of  $F$  requires iterating over the entire data set  $X$ . This takes time  $O(|X|)$  if it assumed for simplicity that  $f$  can be computed in time  $O(1)$ . The crucial feature of this problem is that the set  $Y$  can be accessed in superposition, so that we can

obtain the quadratic Grover speedup in searching over it, but  $X$  is a classical data set which cannot be queried in this way. Thus, we could not use quantum algorithms such those for approximate counting to speed up the evaluation of  $F(y)$ . As a result, the classical runtime of  $O(|X| \cdot |Y|)$  turns into a quantum runtime of  $O(|X| \cdot \sqrt{|Y|})$ . If  $|X|$  is comparable to  $|Y|$ , then this erodes much of the savings from Grover's algorithm. Instead we will use a coreset  $X'$  (with weight function  $w$ ) and replace  $F(y)$  with its approximation  $F_w := \sum_{x \in X'} w(x) f(x, y)$ . This results in a hybrid classical-quantum algorithm for the overall problem. A classical computer needs to examine the original data set  $X$  in order to calculate  $X'$  and then a quantum computer can minimize  $F_w$  in time  $O(\sqrt{|Y|} \cdot |X'|)$ . If  $|X'| \ll |X|$  then this yields a nearly quadratic speedup, and if  $\min_y F_w(y) \approx \min_y F(y)$  then this provides a good approximation to the original problem. Coresets satisfying both of these properties are known in a large number of cases. In some cases the size of  $X'$  will depend only on the level of approximation desired and not on the size of the original data set  $X$ . When this happens, the classical or quantum runtime will not depend on the product of  $|X|$  and  $|Y|^{\frac{1}{2}}$ , but instead on (roughly) their sum. Coresets speed up both classical and quantum algorithms, but they increase the relative quantum speedup by reducing the time spent on tasks where there is no known quantum advantage.

There are three main directions in which this basic example can be modified.

*The Durr-Hoyer minimization algorithm* could be replaced by any other quantum algorithm for minimizing functions, such as adiabatic optimization or QAOA. In almost any such algorithm, either  $F(y)$  or its gradients will need to be evaluated, and the cost of doing so will scale linearly with  $|X|$ . Thus, using a coreset can provide significant savings. These explore more below in Algorithms 1.

**Algorithm 1 (general version)** *Non-adaptive coresets for maximum a posteriori estimation.*

**Inputs:** Data  $X = \{x_1, \dots, x_n\}$ . This algorithm needs the user to specify a function  $f$ , a classical algorithm for generating a coreset (e.g. an approximation algorithm followed by importance sampling) and a quantum optimization algorithm (e.g. Grover, adiabatic, etc.)

**Output:**  $y$ , which is likely to be an exact or approximate solution to (5).

**Algorithm:**

1. Given input  $X$ , use the classical algorithm to construct a coreset  $(X', w)$ .
2. Run the quantum optimization algorithm on  $(X', w)$ .

This “algorithm” is more of a framework than a detailed algorithm. However, it can readily be adapted to hard and relevant optimization problems, such as the following example.



**Algorithm 1.1 (specific version)** *Non-adaptive coresets for k-means clustering.*

**Inputs:** Data points  $x_1, \dots, x_n \in \mathbb{R}^d$ , a number of cluster centers  $k$  and an accuracy parameter  $\epsilon > 0$ .

**Output:** Cluster centers  $y_1, \dots, y_k$  approximately minimizing

$$\sum_{i \in [n]} \min_{j \in [k]} \|x_i - y_j\|^2.$$

**Algorithm:**

1. Use the offline coreset algorithm of [12] to construct a coreset  $(X', w)$  of size  $m = O(\epsilon^{-2} k \log(k) \min(k/\epsilon, d))$ .
2. Use Grover to search for the best clustering of  $(X', w)$ . To make the search space finite, we use [41] to reduce the search space to the  $O(m^{dk})$  possible Voronoi partitions. Given such a partitioning, the cluster center is just the weighted center of those points.

The first step of the algorithm takes time  $n \cdot \text{poly}(k, d, \epsilon^{-1})$  time on the classical computer, while the second step takes the quantum computer time  $m^{dk/2} \cdot \text{poly}(m, k, d)$ , since Grover's algorithm requires  $O(m^{dk/2})$  iterations and each inner loop requires time  $\text{poly}(m, k, d)$ . While classical computers of course could also make use of the coreset and achieve a runtime of  $(n + m^{kd+O(1)}) \cdot \text{poly}(k, d, \epsilon^{-1})$ , the resulting hybrid classical-quantum algorithm achieves nearly a quadratic speedup over the purely classical algorithm that also use coresets. It is important to point out that Algorithm 1.1 is a significant specialization of Algorithm 1 and that many easy variants apply. For example, we could replace Grover's algorithm with a heuristic such as the adiabatic algorithm or QAOA. Or we could replace the Euclidean k-means problem with a generalization known as  $M$ -estimators on metric spaces which can handle more general geometries, as well as having other properties, such as being robust to outliers.

Another large class of variants is to use non-adaptive coresets for the other computational tasks:

**Bayesian inference and saddle-point optimization.** This application is fairly straightforward. A small change would be to  $F(y) := r(y) + \sum_x f(x, y)$ , where  $r(y)$  is a regularizer, perhaps intended to favor simpler models. A bigger change would be to perform Bayesian inference. Bayesian inference can be described as sampling from a distribution  $\pi(y) \propto \exp(-F(y))$ . Here too quantum algorithms can achieve roughly quadratic speedups provably, and heuristic algorithms have been proposed which may have better performance. In each case, quantum speedups are not known for iterating over  $X$ , and so reducing the size of  $X$  would increase the relative quantum speedup. One benefit of sampling over optimization is that the samples output by the quantum computer could be used by the classical computer to adaptively augment the coreset. This idea is explored below in Algorithms 2.

*Remark.* There is a sense in which Algorithm 1 in all its flavors does not use the quantum computer in a very interesting way. Arguably the algorithm is mostly classical, with the quantum computer being used only after sophisticated classical algorithms have reduced the data set to a representative sample. The remaining algorithms will instead use the quantum computer interactively. In many cases the quantum algorithm could also return a  $q$ -sample at little or no extra cost. A  $q$ -sample from a distribution  $\pi$  is defined to be the state  $\sum_y \sqrt{\pi(y)} |y\rangle$ . The idea of a  $q$ -sample was introduced and can have significant advantages over ordinary samples for some applications. In the example's rejection sampling and quantum simulated annealing already returns  $q$ -samples while decohering quantum walks do not.

**Algorithm 2 (general version)** *Adaptive coresets for Bayesian inference.*

**Inputs:** Data  $x_1, \dots, x_n$ . A description of  $\pi_0$ ,  $f$  and a classical algorithm  $\mathcal{A}$  that takes as input  $X$ , a coreset  $(X', w)$  and a set of samples  $y_1, \dots, y_k$  and outputs an updated coreset. A maximum coreset size  $m$ . A quantum algorithm  $\mathcal{B}$  for Bayesian inference.

**Output:** an approximate sample from  $\pi_{\text{posterior}}$ .

- 1: Initialize  $(X', w)$  to be the empty set.
- 2: **for**  $k \in \{1, \dots, m\}$  **do**
- 3:   Use the quantum algorithm  $\mathcal{B}$  to sample  $y_k$  according to the distribution

$$\pi(y_k) = \exp(F_{X',w}(y_k)) / Z_{X',w}, \quad (23)$$

where  $F_{X',w}$  is from (10) and  $Z_{X',w} = \sum_y \exp(F_{X',w}(y))$ .

- 4:   Use the classical algorithm  $\mathcal{A}$  to update the coreset  $(X', w)$ .
- 5: **end for**
- 6: Output  $y_m$ .

Finally, we will see an interactive algorithm with a rigorous performance guarantee. This will address the problem of *saddle-point* optimization, which we can think of as a zero-sum game with one player's strategy set  $X$  described by a classical database and the other player's strategy set  $Y$  accessible in superposition by a quantum computer.

This algorithm yields an answer within  $O(\varepsilon)$  of the true value. To appreciate its efficiency suppose we take  $\varepsilon$  to be constant. Then the classical algorithm sweeps through the entire dataset  $X$  only  $T = O(\log(|X| \cdot |Y|))$  times (with an inner loop of time  $O(T)$ ), and in any reasonable algorithm it would have to do this at least once. Likewise, the quantum algorithm has an inner loop that takes time only  $O(T)$ , and needs to generate only  $O(T)$  samples. For both the classical and the quantum runtime, we are within a factor of  $O(T^2)$  of the best possible time we could expect. (Note that while algorithms for this problem are known with nearly  $1/\varepsilon$  scaling, they do not have the sparsity properties and so cannot be used here.) In general if it used a Grover-type algorithm for the quantum part (specifically rejection sampling), the run-time will be  $O(|X|T^2)$  for the classical computer and  $O(|Y|^{1/2} T^2)$  for the quantum computer.

For these algorithms to be useful we need  $|Y| \gg |X|$ . Otherwise a classical computer could simulate the quantum computer in less time than it would take to read the dataset. In general we would expect the speedups plotted in Fig. 28.

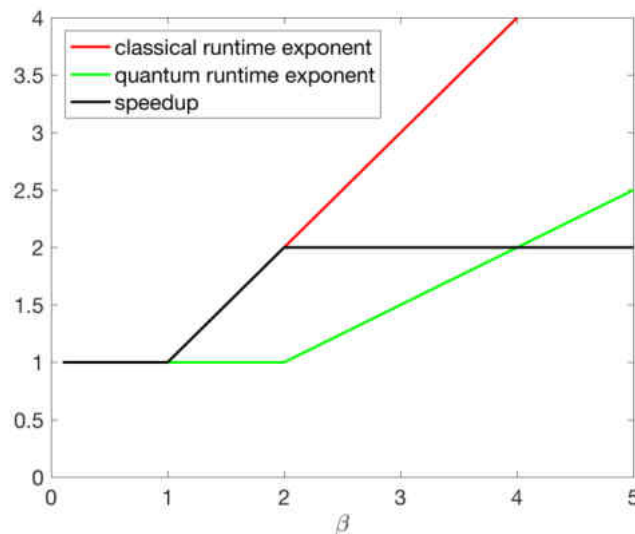


Figure 28. Classical and quantum runtime exponents, as well as speedup, as a function of  $\beta$ .

Assume that  $\alpha = 1$

Other recent work has also proposed quantum algorithms for zero-sum games, using a purely oracle data model. In this setting, they can also obtain a square-root speedup of the search over  $X$ , yielding an overall runtime of  $\tilde{O}(\sqrt{|X|} + \sqrt{|Y|})$ . These works can be viewed as specializations of the SDP algorithms that take advantage of the fact that they are working in the probability simplex. The coreset can be built iteratively using interaction between the classical and quantum processors. First the classical computer produces a coreset  $X'_1$  which the quantum computer uses to produce output  $y_1$ . Then  $y_1$  is used by the classical computer to produce a new coreset  $X'_2$ , which the quantum computer uses to produce output  $y_2$ , and so on for  $r$  rounds. The final answer could either be  $y_r$  or some average of  $y_1, \dots, y_r$ . See Algorithm 3 for details.

**Algorithm 3 (general version)** *Saddle-point optimization (7).*

**Inputs:** A data set  $X = \{x_1, \dots, x_n\}$ , accuracy parameter  $\epsilon > 0$ , and code for a function  $f : X \times Y \mapsto [-1, 1]$ .

**Output:** Distributions  $\theta_Y, \theta_X$  approximately achieving the max or min (respectively) in (7).

1: Let  $T = \lceil \log(|X| \cdot |Y|) / \epsilon^2 \rceil$ .

2: **for**  $t \in \{1, \dots, T\}$  **do**

3:   Use the classical computer to sample  $x_t$  according to

$$\Pr[x_t] \propto \exp \left( -\epsilon \sum_{s < t} f(x_t, y_s) \right). \quad (27)$$

4:   Use a quantum sampling algorithm to sample  $y_t$  according to

$$\Pr[y_t] \propto \exp \left( \epsilon \sum_{s \leq t} f(x_s, y_t) \right). \quad (28)$$

5: **end for**

6: Let  $\theta_X$  be the measure that places weight  $1/T$  on each of  $x_1, \dots, x_T$ .

7: Let  $\theta_Y$  be the measure that places weight  $1/T$  on each of  $y_1, \dots, y_T$ .

The common theme in these algorithms is that there is an outer loop involving  $Y$  and an inner loop involving  $X$ . This outer loop could involve iterating over all elements of  $Y$ , performing a Grover-style search, using adiabatic optimization, or other classical or quantum algorithms. Suppose that in general this outer loop requires  $\tau_{\text{outer}}(Y)$  iterations (e.g.  $|Y|$  for classical brute-force search,  $\sqrt{|Y|}$  for Grover, and so on). Then if the inner loop sums over all of  $X$ , the overall algorithm will require  $O(|X| \tau_{\text{outer}}(Y))$  evaluations of  $f$ . Suppose we have a coreset  $X'$  that can be constructed in time  $\tau_{\text{core}}(X)$ . Then the total (classical + quantum) run-time becomes  $O \left( \underbrace{\tau_{\text{core}}(X)}_{\text{classical}} + \underbrace{|X'| \tau_{\text{outer}}(Y)}_{\text{classical or quantum}} \right)$ . Assume that  $\tau_{\text{core}}(X)$  scales roughly with  $|X|$  while  $|X'|$  is roughly independent of  $|X|$  and is determined instead by the complexity of the model and the desired accuracy. Then we have again replaced a run-time that scales as the product of  $|X|$  and  $\tau_{\text{outer}}(Y)$  with one that scales roughly as their sum. If the run-time is dominated by the complexity of searching over  $Y$  then this will increase the relative quantum speedup.

Suppose for simplicity that  $|X| = n$ ,  $|X'| = O(1)$ ,  $\tau_{\text{core}}(X) = n^\alpha$ , and  $\tau_{\text{outer}}^{\text{classical}}(Y) = n^\beta$ , for some constants  $\alpha, \beta > 0$ . Assume as well that the quantum speedup is quadratic (e.g. based on Grover) so that

$\tau_{\text{outer}}^{\text{quantum}}(Y) = n^{\beta/2}$ . We can summarize the effects of both coresets and classical-vs-quantum computing as follows.

Using coreset?	No	Yes
Classical	$n^{1+\beta}$	$n^\alpha + O(n^\beta)$
Quantum	$n^{1+\beta/2}$	$n^\alpha$ classical and $O(n^{\beta/2})$ quantum

In the lower-right box, the algorithm uses both classical and quantum resources, since the classical computer constructs the coreset in time  $n^\alpha$  and the quantum computer performs the optimization in time  $O(n^{\beta/2})$ . The situation is similar if we replace a simple coreset with an iterative construction. To define the speedup, we say that if the classical and quantum runtimes are  $T_{\text{cl}}, T_q$  respectively then the speedup is  $\log(T_{\text{cl}}) / \log(T_q)$ . Thus “1” means no speedup, “2” is the Grover speedup,  $\infty$  would mean a superpolynomial (e.g. exponential) speedup and  $< 1$  would mean a slowdown. Suppose that  $\alpha = 1$ , since this is often achievable (as we discuss below). The resulting quantum speedups as a function of  $\beta$  are illustrated in Fig. 28.

## Data model

An important departure of this work from much of the quantum machine learning literature is in the choice of input data model. This seemingly mundane issue turns out to be crucial to understanding the utility of many quantum algorithms and will broadly review the role of different data models in quantum algorithms. Existing quantum algorithms have used several different input models. The Standard model is a string of bits  $x = (x_1, \dots, x_n)$  which could be thought of as either as labeling a standard basis state  $|x\rangle$  that is an input to a quantum circuit, or as input to a classical computer which generates a quantum circuit  $C_x$  which is applied to a fixed input. The former interpretation is the standard theoretical model of quantum circuits, while the latter is closer to how quantum computers would be likely to function in practice. Another important model is the Oracle model in which the quantum computer is given access to a unitary  $O$  such that  $O|i, a\rangle = |i, a \oplus x_i\rangle$ . This would arise most naturally if we are given a (classical or quantum) circuit that can compute  $x_i$  given input  $i$ . Other uniquely quantum models exist as well. In the Quantum Data model, the input is given as an arbitrary  $n$ -qubit quantum state  $|\psi\rangle$ . Another model is the Quantum Oracle, meaning black-box access to an  $n$ -qubit unitary  $U$ .

All of these models have been widely used in the quantum algorithms literature. Prominent examples of algorithms using each model are summarized in Table 6.

The hybrid model in this paper works naturally with classical data stored in the same form that would be used for classical algorithms. There is only a minor way in which current quantum computing hardware does not meet its requirements, which is that pulse generators are often slow to reprogram. This makes it easy to repeatedly run the same quantum algorithm but more expensive to modify the gate set either between iterations or during a single run of the quantum computer. However, this latency will have to be improved much more to meet the requirements of fault-tolerant quantum computing. It also appears to be a limitation that is not fundamental but rather applies to current off-the-shelf technology which was originally designed for other tasks.

Not the only one with optimization algorithms that run on current models of hybrid classical/quantum computers. There has been a recent explosion of interest in variational quantum algorithms, such as those for ground states of Hamiltonians, constraint satisfaction problems, learning and other tasks. These algorithms have the advantage of running on near-term hardware with modest requirements for gates, connectivity and qubits. They also make use of a classical computer to run an algorithm such as gradient descent in the outer loop while the quantum computer is used to evaluate the cost function or its gradients in the inner loop. In this way, variational algorithms make use of the longer memory lifetime of the classical computer. By contrast, our algorithms also use the larger storage of the classical computer. Depending on the computational cost of constructing the coreset, the algorithms may also use many more gates from the classical computer.





Table 6. Examples of algorithms using each input model

Input Model	Definition	Examples
Standard	$x = (x_1, \dots, x_n) \in \{0, 1\}^n$	Factoring and other number theory problems. 3-SAT and combinatorial optimization. variational quantum eigensolver
Oracle	$O i, a\rangle =  i, a + x_i\rangle$	OR (Grover search), max, approximate counting NAND tree, collision, [graph] property testing hidden subgroup problem, welded trees
Quantum Data	given state $ \psi\rangle$	quantum Fourier transform SWAP test, Schur transform, state tomography Hamiltonian simulation, linear systems solver learning with quantum examples
Quantum Oracle	given access to unitary $U$ and controlled unitary $C_U$	phase estimation. quantum sensing and process tomography qubitization and singular value transform

In the Quantum Data model, we sometimes assume instead the ability to perform  $V$  and  $V^\dagger$  for some unitary satisfying  $V|0\rangle = |\psi\rangle$ . In the Quantum Oracle model, the controlled unitary  $C_U$  is of the form  $\sum_{t=0}^{T-1} |t\rangle\langle t| \otimes U^t$ . Here we can take  $T=2$  in some cases or can take  $T$  to be exponentially large in other cases.

**Comparison with variational algorithms.** Variational algorithms use a classical outer loop to perform gradient descent on the circuit parameters of a quantum inner loop. These can be extremely general and, in some cases, amount to performing a local search over the set of all short circuits implementable in a particular hardware model. As a result, they often lack the provable guarantees of algorithms. On the other hand, they can be run on even very simple quantum computers and running them can teach us about what we might expect from future quantum hardware. While existing variational algorithms are not designed for a setting with a large classical dataset, there are some connections with the algorithms running the Frank-Wolfe variant of mirror descent on the weight vector of the coreset. This makes the data-reduction approach closer to the variational algorithms that have recently become nearly synonymous with NISQ algorithms. However, there are two key features of our Algorithm 3 that are not suggested by the usual variational formulation: 1) the ansatz and the classical outer loop are structured carefully to present the quantum computer with a very limited subset of the overall data set, and; 2) the output of the quantum computer is usable for a form of stochastic mirror descent without any of the dimension dependence that is seen in general.

**Comparison with stochastic gradient descent.** Many of the problems with handling large datasets are also relevant to classical computers. For this reason, when training continuously parametrized models on large datasets, the standard classical algorithm is not gradient descent but stochastic gradient descent (SGD). One common version of SGD is to sample a single data point at a time and take a gradient step based on that point. We might imagine using this for the inner loop if the outer loop were, say, the Durr-Hoyer minimization algorithm. One can also interpret between this form of SGD and the usual gradient descent with mini-batch gradient descent, which samples a set of  $k \ll n = |X|$  points at a time and uses these for gradient estimates. We use the term “SGD” to refer to this last Mini-batch gradient descent is most directly comparable to using a coreset of size  $k$ . Traditionally, coresets are sampled using some form of importance sampling while mini-batches are often uniformly sampled. However, this is not necessary, and there have been proposals to use importance sampling also to construct mini-batches approach.

The essential difference between SGD and coresets is that coresets are sampled once and then used throughout the optimization, while SGD draws fresh samples for each gradient step. For classical gradient descent, this difference may not be important, or it may favor SGD. However, when used as a subroutine inside a Grover or Durr-Hoyer search, the stochastic noise introduced by SGD can be harmful. Indeed, the Grover speedup is known to vanish when the oracle is stochastic and has a non-negligible chance of being replaced by the identity operator.

Normally Grover search consists of a series of alternating reflections  $R_S R_O R_S R_O R_S R_O \cdots = (R_S R_O)^T$ , where  $R_S$  reflects about the starting state and  $R_O$  is the oracle call. If a single  $R_O$  term is deleted, say in the  $j^{\text{th}}$  position, then this sequence becomes equivalent to  $(R_S R_O)^j (R_O R_S)^{T-j}$ , using  $R_O^2 = I$ . In other words, a single deletion reverses the order of all later rotations. If each  $R_S R_O$  rotates by an angle  $\theta \sim 1/T$  then  $R_O R_S$  rotates by  $-\theta$  and random deletions result in alternating between these two. Overall this random walk will take time  $\sim T^2$  to rotate by an  $\Omega(1)$  angle, thus negating the Grover speedup. If the failure rate is  $\varepsilon \ll 1$  then we can view this as a random walk with step size  $1/T\varepsilon$ , so the total time becomes  $T^2\varepsilon$  (or  $T$ , whichever is greater). Of course, one may consider other strategies, but prove that nothing asymptotically better is possible. This restriction applies only to stochastic oracles. If the oracle produces a coherent superposition of success and failure then the Grover speedup is possible, but in our setting that would require superposition access to the entire dataset. SGD is equivalent to a fault Grover oracle by considering a simple toy model. Suppose we would like to estimate  $\max_{y \in [m]} F(y)$  with  $F(y) = \sum_{x \in [n]} f(x, y)$  and we are promised that each

$f(x, y) \in \{0, 1\}$  and that  $F(y) = \begin{cases} n\varepsilon & y = y_* \\ 0 & \text{otherwise} \end{cases}$ . In other words, there is a subset of  $n\varepsilon$  values of  $x$  for

which  $f(x, y_*) = 1$ . For all other values of  $x, y$ , we have  $f = 0$ . If we use a coreset, then a set of size  $1/\varepsilon$  will have constant probability of hitting a good value of  $x$ . We can then find  $y_*$  with constant probability in time  $\sim \sqrt{m}/\varepsilon$ . When quantum black-box optimization/sampling algorithms rely on computing a function in the inner loop, and that function involves a classical data set, we can use classical data-reduction techniques to reduce the effective size of this set. Without doing so, we could expect many quantum speedups to be significantly weakened, and Grover-type speedups would become effectively useless.

## *Parallel Quantum Simulation of Large Systems on Small NISQ Computers*

Implemented on NISQ machines they allow simulation of quantum systems that are much larger than the computational machine itself. This is achieved by parallelizing the quantum simulation. Cirq and Qiskit code that translate infinite, translationally invariant matrix product state (iMPS) algorithms to finite-depth quantum circuit machines, allowing the representation, optimization and evolution arbitrary one-dimensional systems. Such approaches allow one to concentrate computational resources in the appropriate region of Hilbert space and provide an effective and universal way to simulate quantum systems. They also provide an effective framework to distribute entanglement resources in simulation on noisy intermediate scale quantum (NISQ) computers. Quantum computers such as those of Google, Rigetti, IBM and others implement finite-depth quantum circuits with controllable local two-qubit unitary gates. Currently available NISQ devices are limited by gate fidelity and the resultant restriction of available entanglement resources.

Since the finite-depth quantum circuit may be equivalently described as a tensor network, tensor networks provide a convenient framework with which to distribute entanglement to the useful regions of Hilbert space and to make efficient use of this relatively scarce resource. the implementation of a tensor network on such a NISQ device a Quantum tensor network. There are several advantages to this framework. It fits directly into a broader ecosystem of classical simulation of quantum systems. Indeed, because it is based upon the manipulation of explicitly unitary elements, the quantum circuit provides perhaps the most natural realisation of tensor networks. Canonicalisation at each step in a classical tensor network calculation amounts to reducing the tensors to isometries — a step that is not required in an explicitly unitary realization. Moreover, the remaining elements of unitaries parametrize the tangent space of the variational manifold. Quantum tensor networks can be used to parallelize quantum simulation of systems that are much larger than available NISQ machines. Central to this is dividing the quantum system into a number of sub-elements that are weakly-entangled and can be simulated in parallel on different circuits. The influence of the different regions of the system upon one another can be summarised by an effective state on a much smaller number of quantum bits.

Cirq and Qiskit code for the simplest class of examples — infinite, translationally invariant quantum spin chains. This is a direct translation (mutatis mutandis) of iMPS algorithms to quantum circuit machines. The

remarkably simple circuits revealed below allow the representation of an infinite quantum state, and its optimization and on-line evolution for a given Hamiltonian.

### Parallel quantum simulation across weakly-entangled cuts

To parallelize the simulation on a small NISQ machine, it first identifies partitions of the system where the effect of one partition upon the other can be summarized by a small amount of information. This is achieved by making Schmidt decompositions across the cut:  $|\psi\rangle = \sum_{\alpha=1}^D \lambda^\alpha |\phi_L^\alpha\rangle \langle \phi_R^\alpha|$  where  $|\phi_L^\alpha\rangle$  are an orthonormal set of states to the left of the cut and  $|\phi_R^\alpha\rangle$  the same on the right. The  $\lambda^\alpha$  are known as the Schmidt coefficients and  $D$  the Schmidt rank or bond order. Retaining  $\lambda^\alpha$  only above some threshold value provides a way to compress representations of a quantum state; the MPS construction can be obtained by applying this procedure sequentially along a spin chain.

If an observation is made on the right-hand-side of such a cut, the effect of the quantum state on the left upon the observation can be summarised by just  $D^2$  variables. This same effect can be achieved by an effective state on a spin chain of length  $\log_2 D$  (see Fig. 29), which can be parametrized on the quantum circuit by an  $SU(D^2)$  unitary  $V_L$ .

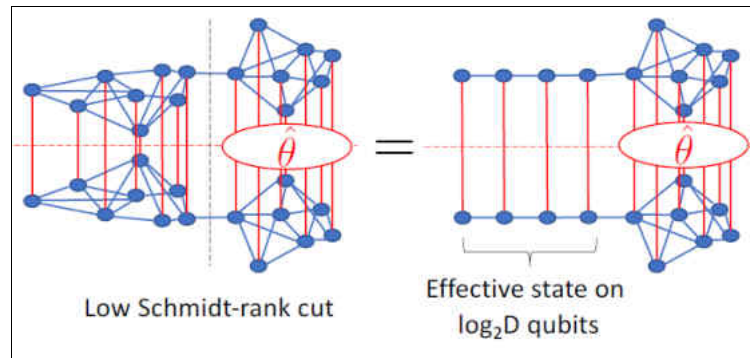


Figure 29. Tensor network for a quantum state that is weakly entangled across a certain partition. This weak entanglement allows parallel simulation of the two partitions of the system. The expectation of an operator located to the right of the partition can be carried out by replacing the state on the left by a state over much fewer spins (the number determined by the entanglement across the cut). The numerical values of correlations in this smaller representation of the left are determined by quantum effects in the full left-hand system, and can be computed in parallel and iterated to consistency

The precise numerical values must be determined by solving a quantum mechanical problem on the left of the system. Similarly, for observations made to the left of the cut, the effect of the right-hand side can be summarized by a unitary  $V_R$ .

This gives a prescription for parallel quantum simulation. Calculations of the quantum wavefunction to the left and right of the cut can be carried out on different quantum circuits or sequentially on the same circuit. The effects of the left partition upon the right partition and vice versa — through the environment unitaries  $V_L$  and  $V_R$  — are iterated to consistency. At each stage of this iteration, measurements must be performed in order to determine  $V_{L/R}$ . The small Schmidt rank of the cut reduces the computational complexity of this process — if we were to do full state tomography, to  $O(\text{poly}(D))$ , but with more sophisticated methods even to  $O(\log(D))$ .

### Parallel simulation with quantum tensor networks

**Representing the state:** A translationally invariant, spin 1/2 MPS state of bond order  $D = 2^N$  can be represented by the infinite circuit shown in Fig. 30 a). Expectations of local operators in this state can be evaluated by the finite circuit shown in Fig. 30 b). The effect of contracting the infinite circuit to the left of the operator is trivial due to the unitarity of  $U \in SU(2D)$  (which automatically encodes the left canonical form

of the related MPS tensor). The contraction to the right is described by the tensor  $V \in SU(D^2)$ , which encodes an effective state over  $N = \log_2 D$  spins and their entanglement with the remaining system to the left-hand side. This unitary is determined self-consistently from  $U$  by the circuit shown in Fig. 30 c).

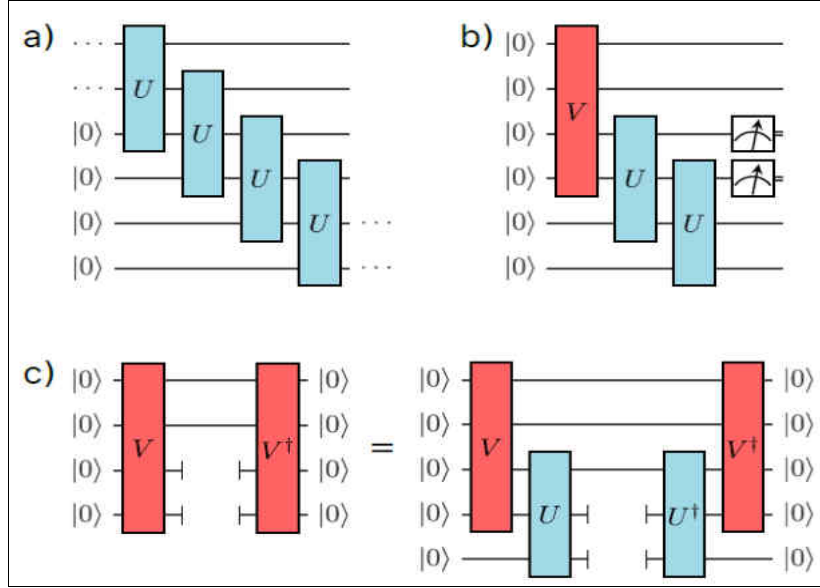


Figure 30. Quantum circuits for translationally invariant states and their local measurement. a) An infinite depth and width quantum circuit representing a translationally invariant state.  $U \in SU(dD)$  with  $d$  the local Hilbert space dimension and  $D = 2^N$  the bond order.  $d = 2$  for spin 1/2 and is used exclusively throughout this section. In these illustrations  $D = 4$ . The circuit acts upon a reference state  $|000\dots\rangle$  at the left of the figure with unitary operators applied sequentially reading left to right. b) Local measurements on this translationally invariant state can be reproduce exactly by the finite circuit shown. The reduced form takes advantage of the unitarity of  $U$ , due to which sites to the left of the observable do not contribute. The environment unitary  $V \equiv V(U) \in SU(D^2)$  summarizes the effect of sites to the right of the observable and describes an effective state over  $N = \log_2 D$  spins. c) The environment unitary  $V(U)$  is the solution of the fixed-point equation shown. This equation is to be interpreted as an equality of the reduced density matrices implied by the free qubit lines. Methods implement this using swap gates. In general,  $V \equiv V(U)$  is not known and must be solved following Fig. 30 c)

The mapping from matrix product state (MPS) to quantum circuits that used automatically embodies much of the variational manifold and its tangent space. The parsimony of this mapping to the quantum circuit suggests that it is the natural home for MPS. The fundamental building block of the circuits depicted in Fig. 30 is the MPS tensor. A tensor of bond order  $D$  and local Hilbert space dimension  $D$  is represented by an  $SU(dD)$  matrix following  $A_{ij}^\sigma = U_{(1 \otimes j), (\sigma \otimes i)}$  as shown in Fig. 31.

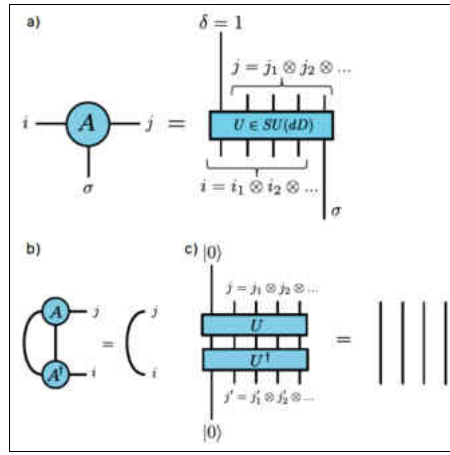


Figure 31. Translation between MPS quantum circuits: a) The translation of an MPS tensor to a quantum circuit. The auxiliary index of bond order  $D$  is created from  $N = \log_2 D$  qubits. b) The canonical form implies that the effect of contracting the MPS to the left is trivial. c) With the mapping of the MPS to a quantum circuit, the unitarity of  $U$  automatically puts the tensor in left canonical form

This translation automatically encodes the left canonical form of the MPS tensor. This follows directly from the unitary property of  $U$ . A classical implementation of an MPS algorithm involves returning the tensors to this form after each step in an algorithm using singular value decompositions — in a quantum algorithm, such a manipulation is not required.

This translation automatically encodes the left canonical form of the MPS tensor. A classical implementation of an MPS algorithm involves returning the tensors to this form after each step in an algorithm using singular value decompositions — in a quantum algorithm, such a manipulation is not required. Moreover, the remaining elements of the unitary encode the tangent space structure to the sub-manifold of states spanned by MPS. These are important in constructing the projected Hamiltonian dynamics. This structure is responsible for the very compact quantum implementation of the time-dependent variational principle shown in Fig. 32.

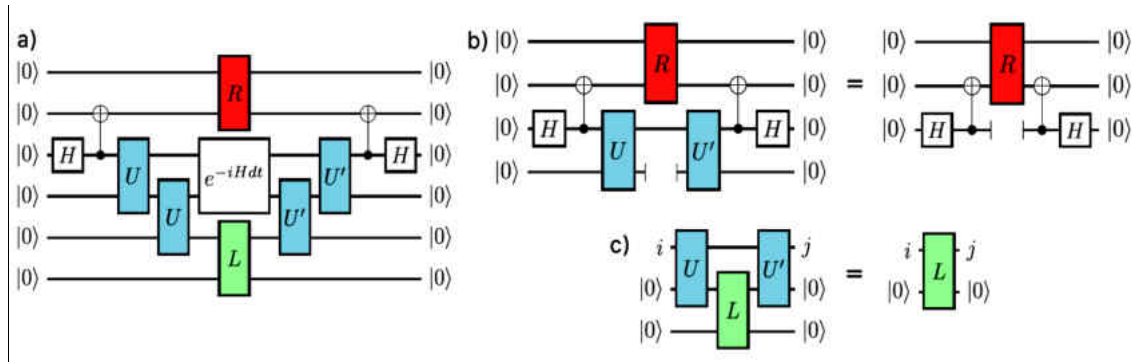


Figure 32. Quantum Implementation of the Time-dependent variational principle. [For simplicity, it depict the above circuits for  $D = 2$ . Higher bond order cases are given in the supplementary materials. a) The unitary  $U_0$  that optimizes the overlap of this circuit with  $|000\dots\rangle$  describes the time evolution of the state described by  $U(t)$  by a time interval  $dt$  under the Hamiltonian  $\mathcal{H}$ , i.e.  $U' = U(t + dt)$ . b) and c) The mixed environment unitaries  $R$  and  $L$  are given by the fixed-point solutions of these circuit equations

[As in Fig.31, these are to be interpreted as an equality of the density matrices implied by free qubit lines.]

**Optimizing the state:** We can find the ground state and the corresponding energy density of translationally invariant Hamiltonians by minimizing the expectation value of the energy. The algorithm mirrors the variational quantum eigensolver. The expectation of the local Hamiltonian is found by measuring the corresponding Pauli strings on the physical qubits (see Fig. 2b)). The result can then be minimized as a function of the ansatz parameters. Updates must be interleaved with updates to the environment,  $V$ , such that it optimizes over valid translationally invariant states.



**Evolving the state:** Perhaps the most compelling feature of this implementation is the ease with which time evolution can be achieved. The simple circuit shown in Fig. 3 a) returns the unitary  $U' = U(t + dt)$  that updates the state encoded by  $U(t)$  to a time  $t + dt$  under evolution with the Hamiltonian  $\mathcal{H}$ . The first variation of this circuit with respect to  $U'$  returns the time-dependent variational principle for iMPS in the form first presented by Haegeman et al. The equivalence uses the automatic encoding of the gauge-fixing of the state to canonical form as well as encoding of the tangent space and its gauge fixing. As in the determination of the best ground state approximation above, the update involves two nested loops; one to find the update  $U'$  and one to find the environment tensors  $L = L(U, U')$  and  $R = R(U, U')$  — both of which are required in this case as the circuit corresponds to the overlap of two different states rather than expectations taken in a given state. A slightly different way of representing these environments in Fig. 3 compared to that employed in Fig. 31.

**Representing the environment.** The simulations presented in the main paper were performed at  $D = 2$ . At  $D = 2$ , the depth required for a full parametrization of the environment is easily accessible, and a minimal parametrization is available. The resulting circuits allow us to calculate important quantities (the Schmidt coefficients, for example) classically, and point the way to natural generalizations for  $D > 2$ .

**A full parametrization of the right environment at  $D = 2$**  The environment - the right fixed point,  $r$ , of the MPS transfer matrix - is an Hermitian, positive definite matrix, with trace 1.  $V$  embeds the Cholesky decomposition of the right environment in a unitary. To obtain the right environment, parametrize  $V$  as shown in Fig. 33.

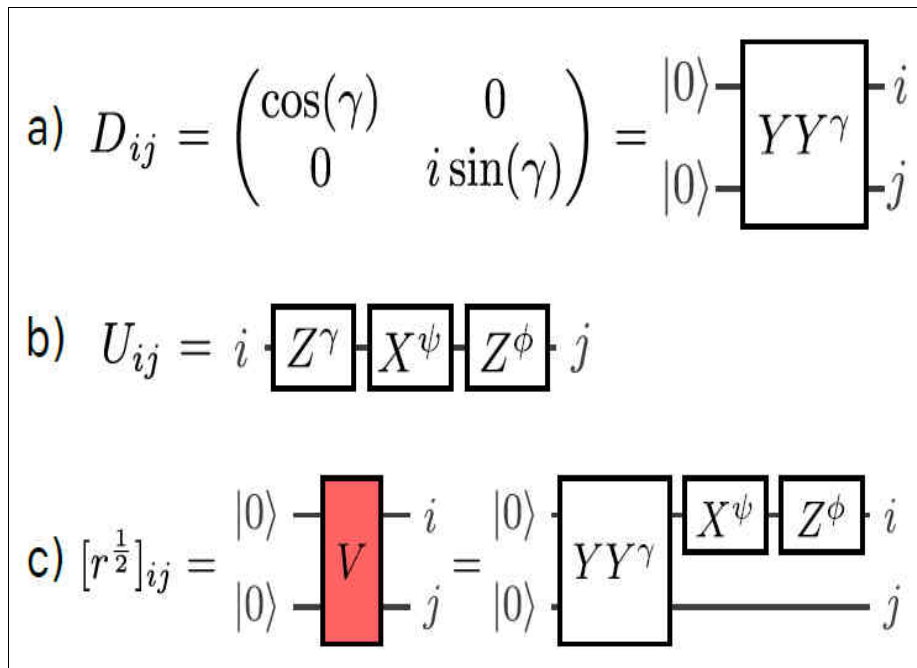


Figure 33. Parametrization of the environment  $V$ , used throughout

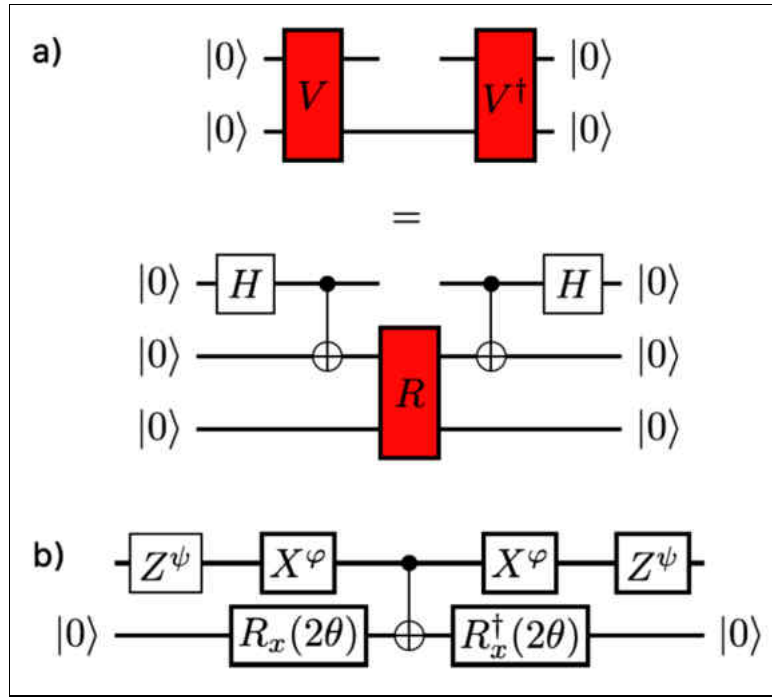


Figure 34. a) The alternative representation of the environment, used in the TDVP circuits and b) the corresponding full parametrization of the unitary  $R$

Since the environment  $r$  has trace 1, the Cholesky factor  $\tau^{1/2}$  has Frobenius norm 1. First consider the matrix in Fig. 33 a). This spans all possible diagonal environment square roots, since  $\text{Tr}(D^\dagger D) = 1, D^\dagger D > 0$ , and  $\cos(\gamma)$  is surjective onto  $[-1, 1]$ , where  $G(\gamma)$  is defined in Fig. 33. Any  $2 \times 2$  Hermitian, unit trace matrix can be written as  $UD(\gamma)^\dagger D(\gamma)U^\dagger$  for some  $U$  and  $\gamma$ . An arbitrary unitary on 1 qubit is shown in Fig. 33 b). Since diagonal matrices commute, the unitary of Fig. 33 c) suffices for  $V$ . This parametrization has some advantages: *i*. It is minimal, depending upon only 3 real parameters; *ii*. The eigenvalues of the environment are classically accessible. The purity, Von Neumann entanglement entropy etc. are all accessible with simple calculations if it knows the parameters of the ansatz.

**Alternative representations of the environment.** When considering the quantum circuits used to implement the time-dependent variational principle, it is convenient to use a different representation of the environment as indicated in Fig. 34 a). This restructuring allows us to trade depth for qubits, and makes manifest the time reversal symmetry of the TDVP circuits. It also allows us to explore non Hermitian tensors variationally. This is crucial in the calculation of the overlap. The parametrization of the tensor  $R$  (Fig. 34 b) retains the benefits of the parametrization of Fig. 33 c), in particular, the eigenvalues of the environment (the square of the Schmidt coefficients across each bond) are:  $\lambda_1 = \cos^2(\theta), \lambda_2 = \sin^2(\theta)$ .

**Optimising the state:** We can find the ground state and the corresponding energy density of translationally invariant Hamiltonians by minimizing the expectation value of the energy. The algorithm mirrors the variational quantum eigensolver. The expectation of the local Hamiltonian is found by measuring the corresponding Pauli strings on the physical qubits (see Fig. 30 b)). The result can then be minimized as a function of the ansatz parameters. Updates must be interleaved with updates to the environment,  $\underline{V}$ , such that we optimize over valid translationally invariant states. In order to determine the environment on the quantum chip it is needed to implement the equality of Fig. 30 from the main paper.

Figure 35 details the quantum circuits required to do so.

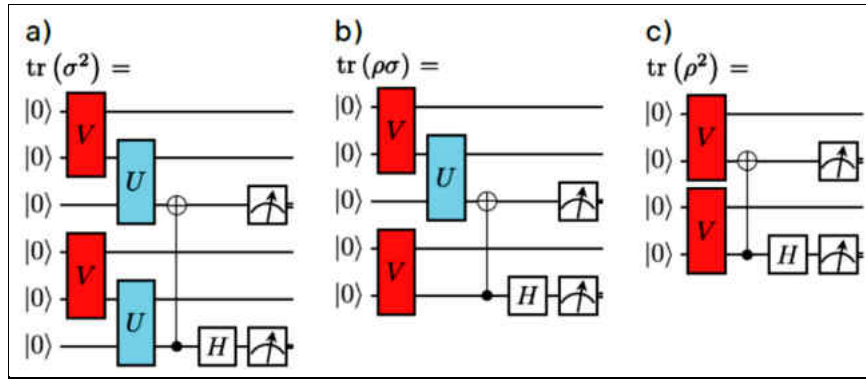


Figure 35. Quantum circuits to find the environment: In order to determine an approximation to the overlap, the fraction of 00 outputs is measured on the measurement symbols

The SWAP test to compare both sides of the equation. The minimal application of this is illustrated by the circuit shown in Fig. 35 b). However, the overlap  $F(\rho, \sigma) = \text{Tr}(\rho\sigma)$  is not necessarily maximized by  $\rho = \sigma$ . In fact,  $F(\rho, \rho) = \text{Tr}(\rho^2)$  is the purity, so that if the reduced density matrices of Fig. 35 b) are mixed. The optimizer will incorrectly try to increase the purity. We circumvent this by using a quantity related to the trace distance  $\text{Tr}(\rho - \sigma)^\dagger (\rho - \sigma)$ , which is minimized at  $\rho = \sigma$ . The circuits required to determine the trace distance are shown in Figs. 35 a) and c).

## Simulation results

We have written Cirq and Qiskit code to implement the quantum circuits shown in Figs 30 and 32.

**PXP model.** Relaxation of few-body quantum systems can strongly depend on the initial state when the system's semiclassical phase space is mixed, i.e., regions of chaotic motion coexist with regular islands. In recent years, there has been much effort to understand the process of thermalization in strongly interacting quantum systems that often lack an obvious semiclassical limit. Time-dependent variational principle (TDVP) allows to systematically derive an effective classical (nonlinear) dynamical system by projecting unitary many-body dynamics onto a manifold of weakly-entangled variational states. Such dynamical systems generally possess mixed phase space. When TDVP errors are small, the mixed phase space leaves a footprint on the exact dynamics of the quantum model. For example, when the system is initialized in a state belonging to a stable periodic orbit or the surrounding regular region, it exhibits persistent many-body quantum revivals.

**A brief overview of time-dependent variational principle (TDVP).** TDVP equations of motion are obtained by extremizing the action  $S = \int dt \mathcal{L}$ ,  $\mathcal{L} = \frac{i}{2}(\langle \dot{\psi} | \dot{\psi} \rangle - \langle \dot{\psi} | \psi \rangle) - \langle \dot{\psi} | H | \psi \rangle$ . Such action yields a set of nonlinear classical equations of motion  $\sum \dot{x}_a \text{Im} \langle \partial_{x_b} \psi | \partial_{x_a} \psi \rangle = -\frac{1}{2} \partial_{x_b} \langle \dot{\psi} | H | \psi \rangle$ , where the variational parameters are real-valued,  $x_i \in \mathbb{R}$ . The equations of motion can also be obtained by minimizing the discrepancy between exact quantum dynamics and its projection onto the variational manifold. At long times, a generic quantum many-body system is eventually expected to develop entanglement that cannot be captured by the MPS ansatz. At this point, the TDVP dynamics  $|\psi(\{x_a(t)\})\rangle$  and exact unitary evolution,  $e^{-iHt} |\psi(\{x_a(0)\})\rangle$ , are expected to strongly disagree. However, the TDVP equations of motion are generically expected to have short periodic trajectories, see the blue line in Fig. 36.

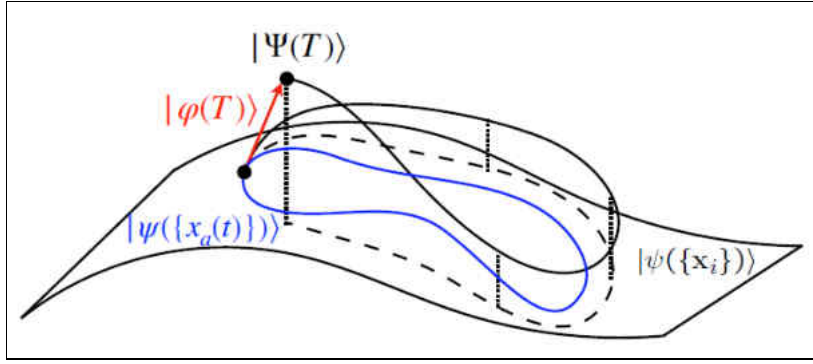


Figure 36. We derive the general bound on the norm of the vector  $j'(t)$  that represents the mismatch between the exact wave function and its TDVP "image" after the TDVP evolution returns to its initial state. The projection of the exact evolution onto TDVP manifold does not coincide with the TDVP trajectory at finite times

If the quantum system were following the TDVP dynamics exactly, that would imply persistent oscillations in local observables and revivals of the many-body quantum fidelity to unity. Due to non-zero leakage, the quantum and TDVP evolution would disagree after a single period. In what follows obtain a lower bound on the many body fidelity, after the time  $T$ , i.e., the period of the trajectory.

To obtain the bound we represent the exact wave function,  $|\psi\rangle$ , that follows the exact evolution according to the Schrodinger equation,  $i\partial_t |\psi\rangle = \mathcal{H}|\psi\rangle$ , as a sum of two terms,  $|\psi(t)\rangle = |\psi(\{x_a(t)\})\rangle + |\varphi(t)\rangle$ , where  $|\psi(\{x_a(t)\})\rangle$  belongs to the variational manifold and is obtained from the TDVP equations of motion, while  $|\varphi(t)\rangle$  is the error vector defined as the difference between these two wave functions, see Fig. 36. Expressing the error vector via  $|\psi\rangle$  and the TDVP wave function, we obtain the following equation of motion:

$$\begin{aligned} \partial_t |\varphi(t)\rangle &= -i\mathcal{H}|\psi(t)\rangle - |\dot{\psi}(\{x_a(t)\})\rangle = -i\mathcal{H}(|\psi(\{x_a(t)\})\rangle + |\varphi(t)\rangle) - x_b(t) |\partial_b \psi(\{x_a(t)\})\rangle \\ &= -i\mathcal{H}|\varphi(t)\rangle - \left[ |\dot{\psi}(\{x_a(t)\})\rangle + i\mathcal{H}|\psi(\{x_a(t)\})\rangle \right], \end{aligned}$$

where explicitly used the Schrodinger equation for  $|\psi(t)\rangle$ . The first term in the last line transports the error forwards in time and does not change the norm of  $|\varphi(t)\rangle$ . The second term describes the change of the error due to mismatch between exact evolution  $|\psi(t)\rangle$  and its TDVP projection,  $|\psi(\{x_a(t)\})\rangle$ , and its norm coincides with the quantum leakage. Calculating the change in the norm of vector obtain

$$\partial_t \langle \varphi | \varphi \rangle = -\langle \varphi | (\partial_t + i\mathcal{H}) |\psi(\{x_a(t)\})\rangle + h.c.$$

where we omit time dependence of  $\varphi$  to simplify notations. Using the triangle and Cauchy-Schwartz inequalities we bound the growth of the norm of the error vector as

$$|\partial_t \langle \varphi | \varphi \rangle| \leq 2 \left| \langle \varphi | (\partial_t + i\mathcal{H}) |\psi(\{x_a(t)\})\rangle \right| \leq 2 \|\varphi\| \|(\partial_t + i\mathcal{H}) |\psi(\{x_a(t)\})\rangle\|,$$

where  $\|\varphi\| = \sqrt{\langle \varphi | \varphi \rangle}$  is the norm of the error vector. Using this notation, we obtain an upper bound on the

rate of increase of the norm of  $|\varphi(t)\rangle$  as  $|\partial_t \|\varphi\|| \leq \Lambda(t) = \|(\partial_t + i\mathcal{H}) |\psi(\{x_a(t)\})\rangle\|$ , where the quantity  $\Lambda(t)$  is the instantaneous geometric error defined above. Now, the triangle inequality can be used to bound the norm of the error vector accumulated during evolution over the finite period of time  $t$ ,  $\|\varphi(t)\| \leq \int d\tau \Lambda(\tau) = I_t$ , by the integrated geometric error  $I_t$ . The Fubini-Study metric or quantum angle can

be defined as  $\gamma(a, b) = \arccos \frac{\|\langle a | b \rangle\|}{\|a\| \|b\|}$ . It is a natural metric on projective space, where pure states are represented by lines through the origin of Hilbert space, reflecting the fact that global phases are not physically meaningful. It is a quotient of the standard Euclidean metric restricted to the unit sphere. The distance  $\gamma$  can be visualized as an angle between the two lines.

The three vectors  $|\psi(t)\rangle$ ,  $|\psi(\{x_a(t)\})\rangle$  and  $|\varphi(t)\rangle$  form sides of a triangle with the angle between the exact quantum wave function and its TDVP approximation equal to the Fubini-Study distance between those two states. Using the law of cosines,

$$\gamma(\psi(\{x_a(t)\}), \psi) \leq \begin{cases} \frac{\pi}{2} & \text{if } I_t > \sqrt{2} \\ \arccos(1 - I_t^2 / 2) & \text{otherwise} \end{cases}.$$

Assuming that we initialized the system on the periodic TDVP trajectory, and taking the time  $t = T$  to be the period of this trajectory, we arrive at the following bound on fidelity,  $\sqrt{F_\psi} \geq 1 - \frac{I_T^2}{2}$ , where

$F_\psi = \left| \langle \psi(\{x_a(T)\}) | \psi(t) \rangle \right|^2 = \left| \langle \psi | e^{-i\mathcal{H}T} | \psi \rangle \right|^2$  and we used the fact that the quantum system is initialized on the TDVP periodic trajectory,  $|\psi(0)\rangle = |\psi(\{x_a(0)\})\rangle = |\psi(\{x_a(T)\})\rangle$ .

In the context of few-body quantum chaos, the strong dependence of quantum relaxation rate on the initial state more often emerges not from unstable periodic orbits (as in quantum scars), but from the phenomenon of mixed phase space. According to Kolmogorov-Arnold-Moser (KAM) theorem, weak deformations of an integrable classical system destroy the regular structure of its phase space only in some regions, leading to a coexistence of regular islands with regions of chaotic motion. When a quantum system is initialized in a wave packet residing predominantly in the mixed region of the phase space, it exhibits much slower relaxation compared to the case when the quantum evolution begins in the chaotic region of the phase space.

The simplest tensor network state is the matrix product state. The manifold of translationally-invariant MPS for a system of size  $L$  with a unit cell of fixed size  $K$  is defined as on Fig. 37 (a).

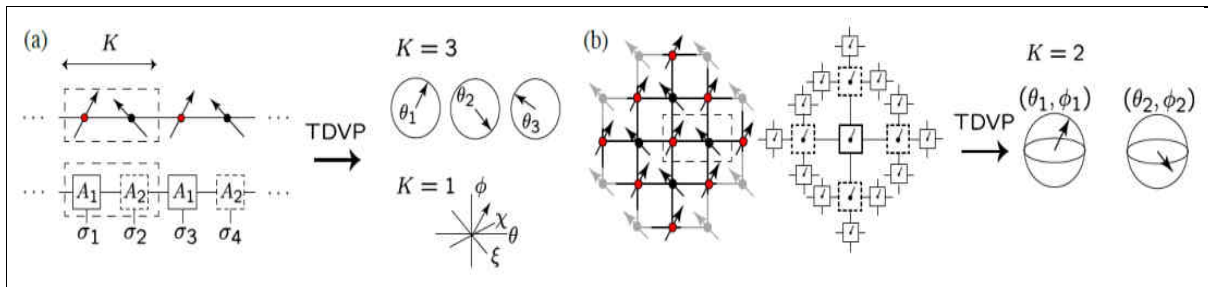


Figure 37. (a) The state of a quantum system that is short-range entangled and translationally invariant with a unit cell of size  $K$  can be described by an MPS with the same size of unit cell. (b) The wavefunction of the square lattice is approximated by a tensor tree state (TTS) in the form of a Cayley tree with connectivity  $C = 4$ . The state with  $K = 2$  unit cell is described by a tree where two different tensors alternate

The MPS state has  $N$  parameters for each of the  $K$  sites within its unit cell. Hence, one needs to specify overall  $NK$  real parameters,  $x_a$  with  $a = 1, \dots, NK$ , to fully fix the state. Below we will use the MPS ansatz restricted to a unit cell of small size and with a small number of variational parameters. These restrictions allow us to obtain analytical equations of motion, making the analysis more transparent. For larger number of parameters, similar analysis could be performed numerically. As another extension, a close relative of the MPS ansatz are the TTS with higher connectivity (see Fig. 1(b)). On the one hand, TTS allow to mimic the topology of two-dimensional and higher-dimensional lattices. On the other hand, the absence of loops still allows for an analytical derivation of equations of motion.



**PXP model.** The PXP model, describing a 1D chain of Rydberg atoms defined by the Hamiltonian:  $\mathcal{H}_{\text{PXP}} = \sum_{i=1}^L P_{i-1} \sigma_i^x P_{i+1}$ , where  $\sigma_x$  is the standard Pauli matrix, and operator  $P_j = (1 - \sigma_j^z)/2$  projects to  $\downarrow$  state on site  $j$ . The projectors encode the kinetic constraint due to the Rydberg blockade: two neighboring atoms are not allowed to be simultaneously excited. Unless explicitly stated otherwise, we work in the thermodynamic limit,  $L \rightarrow \infty$ , and restrict the Hilbert space to spin configurations without two adjacent up spins. Regular regions of phase space are governed by stable periodic trajectories, such as the one in Fig. 38 below.

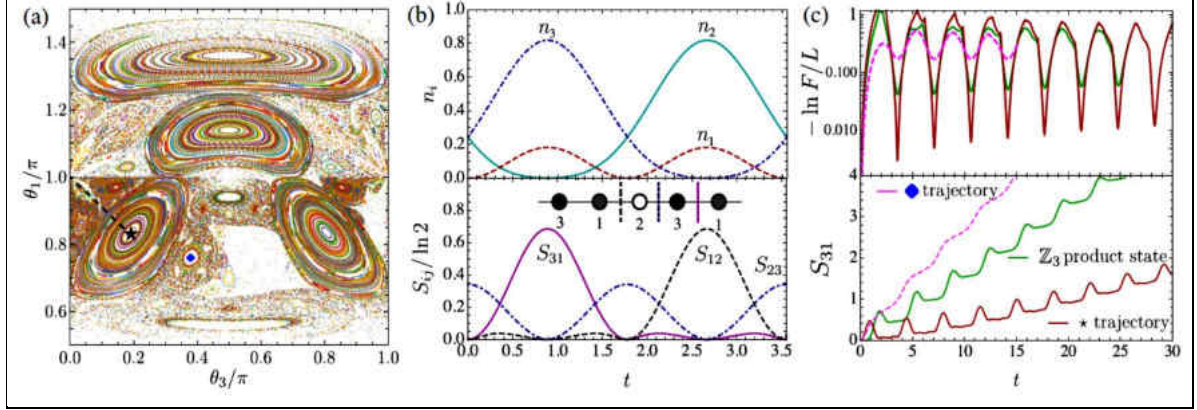


Figure 38. (a) Poincare sections at  $\theta_2 = 0$  for a three-site unit cell reveal mixed phase space with large regular regions; (b) Variational TDVP prediction for the local density of the Rydberg excitation (top panel) and the entanglement entropy (bottom panel) for the shortest-period trajectory; (c) Quantum dynamics for  $|\mathbb{Z}_3\rangle$  initial state and two entangled initial states marked in (a) obtained from iTEBD with  $\lambda = 900$ . Top panel: decay of fidelity per spin has regular minima that correspond to revivals of fidelity in finite size system. Bottom panel: entanglement entropy

These periodic orbits have a vanishing Lyapunov exponent but may be classified according to their “quantum leakage”, i.e., a measure of discrepancy between TDVP and exact quantum dynamics, which intuitively corresponds to the irreversible entanglement growth. On the one hand, short periodic trajectories, with low quantum leakage, generate many-body revivals in quantum dynamics. As a proof of principle, we identify a trajectory with a 3-site unit cell that gives rise to robust revivals, and also generalize the notion of scars to a 2D square lattice of Rydberg atoms. On the other hand, even the trajectories with high leakage out of the variational manifold leave a measurable signature on the quantum system: initializing the quantum system in the vicinity of these trajectories significantly slows down thermalization compared to generic initial conditions.

By introducing a constrained model on the square lattice, inspired by the 1D PXP model:

$$\mathcal{H}_{\text{PXP}} = \sum_{i,j} P_{i-1,j} P_{i,j-1} \sigma_{i,j}^x P_{i+1,j} P_{i,j+1} + \mu_z n_{i,j}, \quad (1)$$

where the pair of indices  $i, j$  is used to label lattice sites. In addition to the PXP term, we have also included a possible perturbation in terms of chemical potential  $\mu_z$ , which couples to the Rydberg excitation density operator  $n = (1 + \sigma^z)/2$ . The presence of four projectors allows a given atom to get excited only if all neighboring atoms are in the ground state. Similarly, to the PXP model in 1D, we will restrict the attention to the largest connected subspace of the Hilbert space, where no adjacent atoms are excited. When detuning  $\mu_z \neq 0$ , the dynamics generated by Eq. (1) takes place on constant energy surfaces in four dimensional space. In order to visualize such dynamics, we fix  $\theta_1$  variable and show the resulting Poincare sections in Fig. 39 (a).

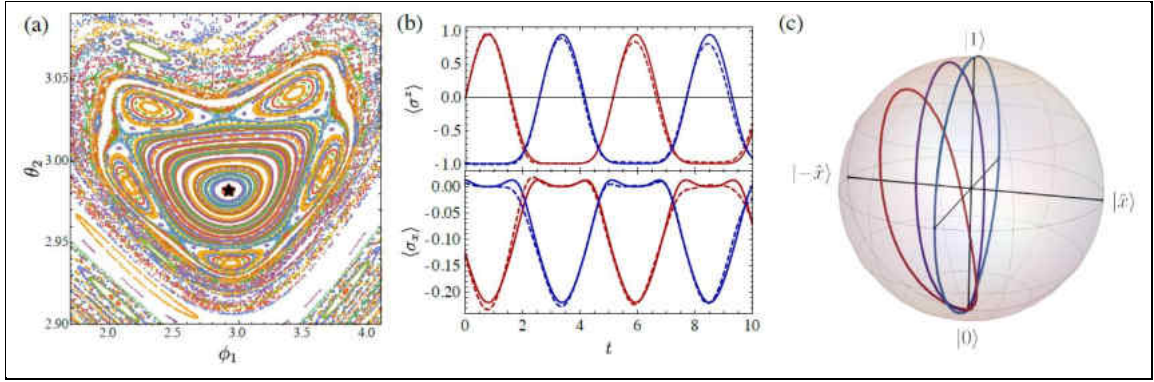


Figure 39. (a) Poincare section for  $\mu_z = 0.225$  reveals mixed phase space with large regular regions.

(b) Comparison between TDVP predictions for the dynamics of local observables ( $\mu_z = 0.225$ ) on the periodic trajectory (solid lines) and the exact dynamics of a  $4 \times 4$  lattice with periodic boundary conditions. (c) Bloch sphere visualization of the individual spin dynamics on the TDVP trajectory. The trajectory on the big meridian corresponds to the case  $\mu_z = 0$ . Non-zero chemical potential  $\mu_z = 0.225, 0.65$  causes a progressive tilting of the trajectory and the development of a knot around the south pole

Variational dynamics of the PXP model and its deformations, studied above, revealed mixed phase space with multiple stable orbits. The Lyapunov exponent, which is often used to quantify chaos in TDVP and semiclassical dynamics, vanishes for all these orbits and thus cannot be used to distinguish them. Quantum leakage can be used instead as a heuristic for distinguishing trajectories that give rise to revivals in exact quantum dynamics.

We have chosen optimization and time evolution of the transverse field Ising model, and Poincare sections of the dynamics of the PXP Hamiltonian as illustrative examples.

Example: PXP model simulation. Figure 40 demonstrates the results of running the quantum time-dependent variational principle code on the PXP model.

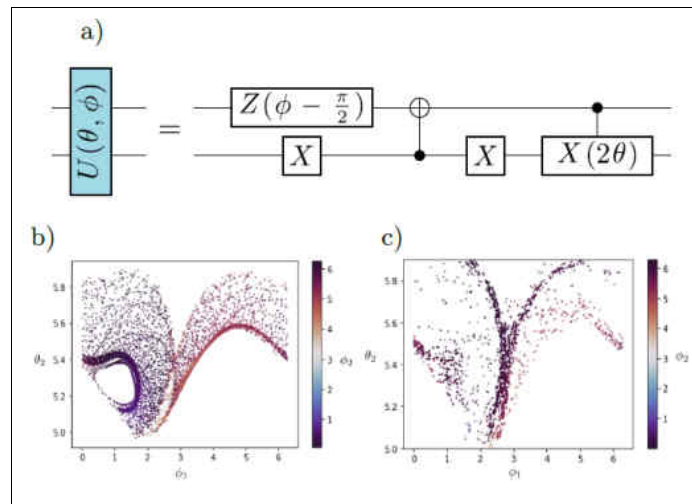


Fig. 40. Many Body Scars in the PXP model: The Hamiltonian  $\mathcal{H} = \sum_n (1 - \hat{\sigma}_{n-1}^z) \hat{\sigma}_n^x (1 - \hat{\sigma}_{n+1}^z)$ , first posited to describe the results of quantum simulations using Rydberg atoms, displays a curious property known as many-body scarring, whereby from certain starting states, persistent oscillations that can be described with a low bond-order MPS are found. These are amenable to study on a NISQ machine using the quantum time-dependent variational principle of Fig 32. a) A simple set of 2-site periodic states at bond order  $D = 2$  are parametrized by circuits with just 2 parameters per site, so 4 in total. b) A partial Poincare section through the plane  $\theta_1 = 0.9$  produced from a classical simulation using the matrix product state equations of motion presented. c) The same Poincare map produced simulating the quantum time-dependent variational principle in Cirq

Such plots have advantages for NISQ machines as they permit the use of error mitigation schemes. To produce Fig. 39 we have used post-selection over energy to discard erroneous points in the Poincare map. This mitigates both integration errors due the finite time-step, stochastic optimization, and potentially errors due to finite gate fidelity. Structures within the Poincare map, while distorted by errors, are nevertheless robust to them.

Example: *The transverse field Ising model*. The results of running this code in simulation on Google's Cirq simulator are shown in Fig. 41. The properties of the transverse field Ising model are well understood. The Loschmidt echo (fidelity of the time-evolved wavefunction with the initial wavefunction) reveals a dynamical phase transition which provides a non-trivial test for our simulation. The main findings are as follows:

- I. When run without gate errors and complete representation of the unitaries  $U$ ,  $V$ ,  $L$ , and  $R$ , the code precisely reproduces the optimum iMPS and its time evolution using the time-dependent variational principle.
- II. Factorizations of the unitaries reduce the fidelity of our results. These are systematically improved as the depth or expressibility of the ansatz is increased. Full parametrizations of the unitaries reproduce classical MPS results exactly.
- III. Going from representing, to optimising, to time evolving states places increasing demands upon circuit depth and width. Accurate results require increasingly deep factorization of the unitaries, and suffer increasingly adverse effects of gate errors.

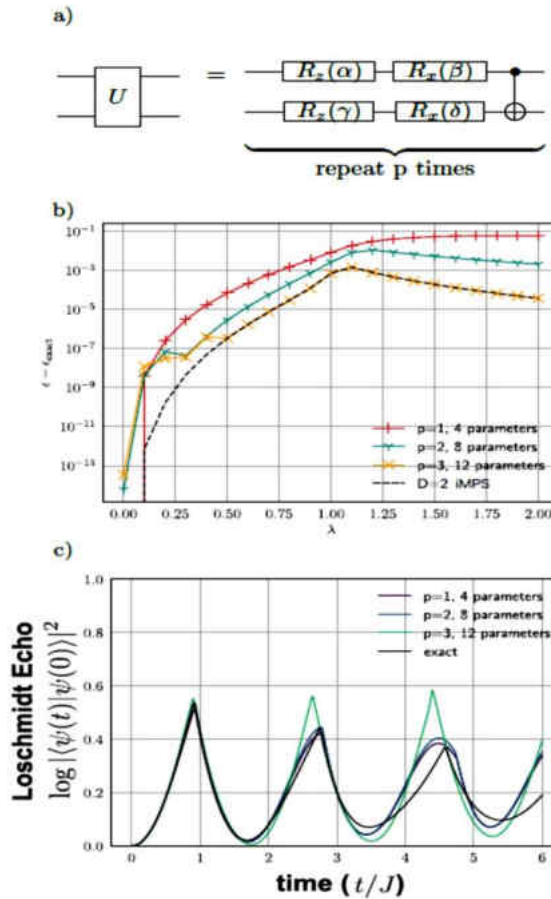


Figure 41. Results of simulating the transverse field Ising model: The Hamiltonian  $\mathcal{H} = \sum_n [\hat{\sigma}_n^x \hat{\sigma}_{n+1}^z + \lambda \hat{\sigma}_n^x]$  is studied with a bond order  $D = 2$  quantum matrix product state. a) The  $SU(4)$  unitaries  $U$  and  $V$  are compiled to the circuit as shown. The parameter  $p$  is varied to increase the accuracy. Although more efficient parametrizations exist for 2 qubit unitaries, as well as circuits more specifically tuned to this problem, we choose a generic circuit. It is readily extendible to higher bond orders. b) The

*optimum state is found using the circuits depicted in Fig. 30. The energy of this state is a better approximation to the true ground state energy as the depth of parametrization increases and converges to that obtained in a conventional MPS algorithm. Note that at  $\lambda = 0$  the Hamiltonian is optimized by a product state, which is captured perfectly with  $p = 1$ . c) Transverse field Ising model displays dynamical phase transitions in the Loschmidt echo*

These are revealed in the simulated runs of the quantum time-dependent variational principle embodied by the circuits in Fig. 32. More accurate circuits are required to obtain good agreement. The results indicated as exact in the above are exact analytical results. It reproduces the Poincare maps. Such plots have advantages for NISQ machines as they permit the use of error mitigation schemes. To produce Fig. 40 have used post-selection over energy to discard erroneous points in the Poincare map. This mitigates both integration errors due the finite time-step, stochastic optimization, and potentially errors due to finite gate fidelity. Structures within the Poincare map, while distorted by errors, are nevertheless robust to them.

## Conclusion

For quantum black-box optimization/sampling algorithms rely on computing a function in the inner loop, and that function involves a classical data set, we can use classical data-reduction techniques to reduce the effective size of this set. Without doing so, it could expect many quantum speedups to be significantly weakened, and Grover-type speedups would become effectively useless. How to design quantum algorithms for machine learning tasks as possible to the power of quantum computers, we might hope that  $BQP=PSPACE$ . Even in this case, we would still have the problem that near-term quantum computers will have a small number of qubits (say  $n$ ) and because of decoherence will not able to run many gates (say  $T$ ). Thus, such a quantum computer could only handle  $O(T)$  pieces of classical data. Without data-reduction techniques, fitting a model to a large data set would not obviously be sped up by a small quantum computer, even with the assumption that  $BQP=PSPACE$ . Conversely, this assumption on the power of quantum computers could be a good place to look for new algorithms, since if quantum computers cannot help in this model, they will never be useful [1].

## References

1. Aram W. Harrow Small quantum computers and large classical data sets // arXiv:2004.00026v1 [quant-ph] 31 Mar 2020.
2. F. Barratt et al. Parallel Quantum Simulation of Large Systems on Small NISQ Computers // 2003.12087v1 [quant-ph] 26 Mar 2020.
3. Van den Brink R. F.M. Vision on Next Level Quantum Software Tooling // Proc. The 10<sup>th</sup> Intern. Conf. Computational Logics, Programming, Tools, and Benchmarking. 2019. — Venice, May 5-9, Italy. — Pp. 16-23.
4. Hadfield S.T. Quantum Algorithms for Scientific Computing and Approximate Optimization. - Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Graduate School of Arts and Sciences. - COLUMBIA UNIVERSITY. — 2018.
5. LaRose R. Overview and Comparison of Gate Level Quantum Software Platforms // arXiv:1807.02500v1 [quant-ph] 6 Jul 2018.
6. Khatri S. Quantum-assisted quantum compiling // aXiv:1807.00800v5 [quant-ph] 7 May 2019.
7. Leo Zhou, Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Device // arXiv:1812.01041v2 [quant-ph] 2019]; The “parameter shift rule” in the larger context of hybrid optimization.
8. Schuld M. Evaluating analytic gradients on quantum hardware // arXiv:1811.11184v1 [quant-ph] 27 Nov 2018.
9. Leo Zhou, Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Device // arXiv:1812.01041v2 [quant-ph] 2019.
10. Schuld M. Evaluating analytic gradients on quantum hardware // arXiv:1811.11184v1 [quant-ph] 27 Nov 2018.

11. Bergholm V. PennyLane: Automatic differentiation of hybrid quantum-classical computations // arXiv: 1811.04968v3 [quant-ph] 14 Feb 2020.
12. Gilyén A., Arunachalam S., Wiebe N. Optimizing quantum optimization algorithms via faster quantum gradient computation // In book: Proc. of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms. – Springer Verlag. — 2019. — Pp.1425-1444. — [arXiv:1711.00465v3 [quant-ph] 17 Apr 2018].
13. Gilyén A. Quantum singular value transformation & its algorithmic applications. — Institute for Logic, Language and Computation Universiteit van Amsterdam. ILLC Dissertation Series DS-2019-03. — 2019.
14. Cornelissen A.J. Quantum gradient estimation and its application to quantum reinforcement learning. — Master thesis. Delft University of Technology. — 2018.
15. Masaya Watabe, Quantum Circuit Parameters Learning with Gradient Descent Using Backpropagation // [quant-ph] 1910.14266. October, 2019.
16. Harrow A.W. Small quantum computers and large classical data sets // arXiv:2004.00026v1 [quant-ph] 31 Mar 2020.
17. Michailidis A. et al. Slow quantum thermalization and many-body revivals from mixed phase space // 1905.08564et al. v2 [quant-ph] 21 Jan 2020.