


ORIGINAL RESEARCH

Psitrum: An open source simulator for universal quantum computers

Mohammed Alghadeer¹  | Eid Aldawsari² | Raja Selvarajan³ | Khaled Alutaibi² | Sabre Kais³ | Fahhad H. Alharbi^{4,5}

¹Department of Physics, Clarendon Laboratory, University of Oxford, Oxford, UK

²Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

³Department of Chemistry, Department of Physics and Astronomy, and Purdue Quantum Science and Engineering Institute, Purdue University, West Lafayette, Indiana, USA

⁴Department of Physics, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

⁵Department of Electrical Engineering, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

Correspondence

Mohammed Alghadeer, Department of Physics, Clarendon Laboratory, University of Oxford, Oxford OX1 3PU, UK.

Email: mohammed.ghadeer@physics.ox.ac.uk

Funding information

National Science Foundation, Grant/Award Number: 1955907

Abstract

Quantum computing is a radical new paradigm for a technology that is capable to revolutionise information processing. Simulators of universal quantum computer are important for understanding the basic principles and operations of the current noisy intermediate-scale quantum processors, and for building in future fault-tolerant quantum computers. As next-generation quantum technologies continue to advance, it is crucial to address the impact on education and training in quantum physics. The emergence of new industries driven by progress in quantum computing and simulation will create a demand for a specialised quantum workforce. In response to these challenges, the authors present Psitrum, an open-source simulator for universal quantum computers. Psitrum serves as a powerful educational and research tool, enabling a diverse range of stakeholders to understand the fundamental principles and operations of quantum systems. By offering a comprehensive platform for emulating and debugging quantum algorithms through quantum circuits, Psitrum aids in the exploration and analysis of various quantum applications using both MATLAB and MATLAB application programming interface to use the software on other platforms. Psitrum software and source codes are fully available at GitHub.

KEYWORDS

quantum computing, quantum information, quantum noise

1 | INTRODUCTION

Quantum computation is a radical new candidate for a technology that is capable to make a paradigm shift in information processing [1]. Quantum computers are now a reality with available quantum testbeds and variety of quantum algorithms

[2]. Computation based on quantum algorithms have proved to be more efficient in processing information and solving wide range of complex problems [3, 4]. Quantum simulators can be designed using quantum algorithms represented by quantum circuits and based on mathematical unitary operations [5]. In this sense, simulators are special purpose quantum circuits

Abbreviations: API, application programming interface; CTM, classical Turing machine; DJ, Deutsch–Jozsa; GUI, graphical user interface; M, number of stages (related to the quantum circuit simulation); N, number of qubits (related to the quantum circuit simulation); NISQ, noisy intermediate-scale quantum; QC, quantum computer; QTM, quantum Turing machine; SDKs, software development kits; VQE, variational quantum eigensolver.

This is an open access article under the terms of the [Creative Commons Attribution-NonCommercial-NoDerivs](https://creativecommons.org/licenses/by-nc-nd/4.0/) License, which permits use and distribution in any medium, provided the original work is properly cited, the use is non-commercial and no modifications or adaptations are made.

© 2024 The Author(s). *IET Quantum Communication* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.

designed to provide insight about specific physical problems. Variety of universal logic gates based on quantum mechanics can be combined to provide very powerful computing features [6]. Such simulators permit understanding quantum systems that are challenging to study in laboratories and impossible to model with the most powerful supercomputers [7, 8].

Universal quantum simulators are based on a QC proposed by Yuri Manin in 1980 [9] and Richard Feynman in 1982 [10]. Feynman showed that a Classical Turing Machine would not be able to simulate complex quantum systems because of the huge amount of information contained with exponential increase in the required size of computational bits, while his hypothetical universal QC can mimic many quantum effects, such as superposition and entanglement, which allows to efficiently simulate quantum systems [11]. Quantum properties have shown to significantly enhance simulation power and computational speed when implemented in quantum algorithms [12, 13].

Quantum simulators may be constructed with generally programmable quantum computers [14], which would be capable for solving a wider class of problems using both classical and quantum algorithms. The later can be defined as a finite sequence of steps for solving a problem in which each step can be executed on a QC. [5, 15]. Instead, a QC simulator can be implemented on the currently available classical hardware to mimic the operation of a real QC, but with limited number of quantum bits (qubits) [16]. Such simulator allows to test and implement quantum algorithms for many applications even before a fully-controllable QC exists [17]. Quantum simulators are not to be confused with simulations of quantum computation on classical hardware which we call a QC simulator.

A universal simulator of quantum circuits can be implemented based on David Deutsch model for a Quantum Turing Machine (QTM) [18], which provides a simple model that captures all the power of quantum computation. Therefore, any quantum algorithm can be expressed formally as a particular QTM. The practical equivalent model is a quantum circuit defined as a quantum algorithm implemented on a gate-

model based QC with special logic gates. In such circuits, only matrix multiplication and tensor products are the advanced mathematical operations that are used. QTM can be related to classical and probabilistic Turing machines in a framework based on transition matrices. As shown by Lance Fortnow [19], a matrix can be specified whose product with the matrix representing a classical or probabilistic machine provides the quantum probability matrix describing the quantum circuit.

Most of the existing quantum computers are still noisy and located in research labs [20]. In addition, the hardware and maintenance of such systems are expensive and not optimised yet [21]. Hence, only limited public access to these computers is available. On the other hand, QC simulators can be useful to overcome such problems. A QC simulator is a software program which imitates the functionality of a QC using classical hardware [22], which allows to design, run and test quantum algorithms. There are many QC simulators available to the public [23]. They differ in purpose, language, size, complexity, performance and technical-based type (e.g. toolkits) [24]. Companies working in this field (e.g. IBM, Microsoft, Rigetti, Google, and ETH Zurich) are creating full-stack libraries for universal quantum computers with useful simulation tools [24–26]. Most of these tools are not software, but software development kits (SDKs) or frameworks, including Qiskit [27], LIQUi [28], ProjectQ [29], Cirq [30], QX [23] and Quantum-sim simulator [31]. Other simulators focus in enhancing some aspects, such as qHiPSTER from Intel [32] that takes maximum advantage of multi-core and multi-nodes architectures, and QuEST [33] which is a multithreaded, distributed and GPU-accelerated simulator. There are few software that allow the user to graphically design a quantum circuit and test it without writing a programming code and most of them are web-based, such as IBM Quantum [34] and Quirk [35]. Details and features among universal quantum simulators are given in Table 1.

Many QC simulators focus on increasing the number of qubits and simulation speed by using different software tools in order to enable simulating real world quantum algorithms and use cases that can benefit from quantum computing. For

TABLE 1 Comparative analysis between general-purpose quantum simulators.

	Psitrum	Qiskit [27]	Cirq [30]	QuEST [33]	ProjectQ [29]	qHiPSTER [32]	Quirk [35]
Platform	MATLAB/API	Python	Python	C++/Python	Python	C++	Web-based
Qubit simulation limit	Medium	Medium	Medium	High	Medium	High	Medium
Noise simulation	Yes	Yes	Yes	Yes	Yes	No	No
Visualisation tools	Yes	Yes	Limited	No	Limited	No	Yes
Educational features	Extensive	Extensive	Yes	No	Yes	No	Yes
GPU acceleration	Yes	No	No	Yes	No	Yes	No
Multi-core support	Yes	No	No	Yes	Yes	Yes	No
Unique features	User-friendly GUI/API	Toolkit for developers	Google services integration	High-performance	Flexible SDK	High performance	Interactive learning

example, Quantsim implemented several optimisations to enable a full density matrix simulation of surface codes in order to evaluate its resilience to noise and whether it can benefit current quantum computing systems to achieve fault-tolerant quantum computation [31]. The challenge here is the exponential increase in the dimension of available quantum states with number of qubits [36, 37]. Furthermore, there are serious shortcomings in the development of abstractions and visualisations of the QC simulation problem. One of these problems is tracing the state of each qubit after each unitary quantum operation [38]. Another shortcoming is in providing useful probabilistic visualisations of the resulted quantum states [39]. In this work, we address these issues with many visualisations tools. There are other shortcomings that need to be resolved later, such as using different simulation methods for visualising quantum states, such as using Feynman path integral formulation [40] and using tensor networks [41].

Quantum computer simulators are important for understanding the operation noisy intermediate-scale quantum (NISQ) processors, and for building future quantum computers. Current and near-term NISQ computers are limited by the presence of different types of quantum noise that are still too large to allow solving relevant scientific problems. One main source of noise is readout errors that occur during measurements [42, 43]. They typically prevent reading the correct state of qubits, such as reading zero while the correct state is one and vice versa. Another important source of quantum noise is gate errors, which can be classified into coherent and incoherent noise [44, 45]. Coherent noise preserve state purity of quantum systems and result in miscalibration in control parameters [46]. Incoherent noise can be modelled as coherent noise with stochastic varying control parameters. This allows to convert coherent errors into incoherent errors through randomised compiling [47, 48]. Incoherent noise are relatively easier to handle because they can be modelled as a process that entangles the quantum system with its environment. One very important class of Incoherent noise includes depolarising channels.

In the realm of quantum education, the availability of versatile tools is crucial for effective teaching and learning. Psitrum offers a significant advantage by providing compatibility with both MATLAB and MATLAB Application Programming Interface (API). This compatibility expands the reach of Psitrum, allowing users to leverage the software on diverse platforms and making it accessible to a broader audience. The utilisation of MATLAB and MATLAB API enhances the educational value of Psitrum, as these widely-used tools enable seamless integration with existing educational materials, workflows, and programming environments. By incorporating Psitrum's capabilities into quantum education, educators can harness the power of MATLAB to deliver comprehensive and immersive learning experiences to students, fostering a deeper understanding of quantum principles and algorithms.

In this work, therefore, we present Psitrum—a universal gate-model based QC simulator implemented on user's local

hardware. The simulator allows to emulate and debug quantum algorithms in form of quantum circuits for many applications with the choice of adding quantum noise that limit coherence of quantum circuits. Psitrum allows to keep track of quantum operations and provides variety of visualisation tools. The simulator allows to trace out all possible quantum states at each stage M of an N -qubit implemented quantum circuit. The design of Psitrum is flexible and allows the user later to add more quantum gates and variety of noise modules.

2 | PSITRUM: AN EDUCATIONAL AND RESEARCH TOOL

Over the time, Psitrum has evolved to be both educational and research tool. Initially, the aim was to develop a tool that can conduct quantum computing experiments using laptops and personal computers with a user friendly interface and more control options than what is allowed by common tools at the time. To handle computations involving a larger number of qubits, efficient programming and a numerical computing platform equipped with a robust algebra library and adept sparsity handling are crucial. There are many options; but the main ones are MATLAB and Python. Certainly, each one has advantages and disadvantages when compared to the other. However, MATLAB extends its compatibility to Python users via 'MATLAB Engine API', which is a package to call and utilise MATLAB in python and other languages. Furthermore, MathWorks, which is the owner of MATLAB, is trying to make MATLAB classroom more user friendly for students. This became very handy to introduce quantum computing to undergraduate students in tutorials and workshops conducted at King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia. The educational value of the code became very apparent then as it allowed demonstrating and simulating quantum experiments for students. According to MathWorks datasheet, more than 6500 universities use MATLAB and Simulink (MATLAB derivative and compatible) for both educational and research purposes [49, 50] with more than 5 million users [49].

Quantum physics is often perceived as one of the more challenging subjects for undergraduate students [51–55]. This perception can be attributed to both intrinsic and extrinsic factors. Intrinsically, quantum physics is somewhat abstract and often counter-intuitive. Extrinsically, its typical placement after several classical physics courses can lead to misconceptions rooted in previously established classical frameworks. Several modifications and solutions have been proposed to address these challenges [53–57]. One of the key elements in these solutions is visualisation of quantum phenomena [53, 54, 57]. In Psitrum, visualisation is an essential constituent with a simple and friendly graphical user interface (GUI). The dynamic adjustability of parameters allows users, whether students or researchers, to directly experience quantum effects, promoting deeper understanding and more accurate interpretations.

For researchers, Psitrum utilises the rich algebra toolboxes and sparsity control in MATLAB to run experiments up to 15 qubit in normal personal laptops, with comprehensive examples presented in Section 5. This can be scaled up using more computing power and utilising MATLAB parallelisation and graphics processing unit capabilities. However, using personal computers is very handy and should allow the users to conduct more experiments and calculations with a reasonable degree of control.

Psitrum is made open through via GitHub. Since its launch, it has gained positive feedback from the quantum sciences community, evidenced by a good number of downloads, comments and contacts. Such engagement underscores the software's practicality and its resonance with users looking for a reliable quantum computing tool.

3 | SOFTWARE STRUCTURE

In this work, we use MATLAB to build Psitrum based on universal quantum gates. A set of single- and multi-qubits gates is defined as the basic building blocks of Psitrum. Any other arbitrary multi-qubits operation can be simply implemented by applying successive tensor products and multiplications of the basic gates. It is sufficient to implement these specific gates as a sequence of arithmetic operations on the input state vector in order to implement any quantum logic operation [58]. The definition of the basic set of quantum gates is presented in section of additional information below. In addition, MATLAB is a well-known software environment that can handle and manipulate matrices in form of unitary operations very efficiently [59]. Psitrum calculates the resulting circuit matrix of any quantum circuit and, in parallel, the software calculates the density matrix as well for applying noise models on the simulation problem. Both the circuit and density matrices are usually very Sparse matrices and, for large circuits, the Sparse packages provided by MATLAB makes the simulation problem very efficient in which it subsequently reduce the memory size requirement and speedup the simulation up to a reasonable number of qubits. While this software was built using MATLAB, Psitrum is a toolbox that can also be used externally in other platforms, such as python, by using MATLAB APIs.

3.1 | Basic operation

For a given quantum circuit and an initial vector state, Psitrum calculates the algorithm matrix, the density matrix and the output quantum states which are used to provide useful visualisations. The simulation process of Psitrum starts by replacing each quantum gate by its unitary matrix. Next, the different M stages are combined by applying N-1 tensor products and M-1 matrix multiplications. Then, the output vector state of N qubits is calculated by multiplying the algorithm matrix by the initialised input vector state of the qubits.

The state of an elementary storage unit of a QC is a qubit, described by a two-dimensional vector of Euclidean length one. The normalised state $|\Phi\rangle$ of a qubit can be written as a linear superposition of two orthogonal basis $|0\rangle$ and $|1\rangle$:

$$|\Phi\rangle = a_0|0\rangle + a_1|1\rangle \quad (1)$$

where a_0 and a_1 are complex numbers that satisfy the normalisation condition $|a_0|^2 + |a_1|^2 = 1$. In general, the normalised state $|\Psi\rangle$ of N qubits is accordingly described by 2^N dimensional unit vector:

$$|\Psi\rangle = a_{(0\dots00)}|0\dots00\rangle + a_{(0\dots01)}|0\dots01\rangle + \dots a_{(1\dots10)}|1\dots10\rangle + a_{(1\dots11)}|1\dots11\rangle \quad (2)$$

where now the complex coefficients must satisfy the normalisation condition:

$$\sum_{n=0}^{2^N-1} |a_n|^2 = 1 \quad (3)$$

where $a_0 = a_{(0\dots00)}$, $a_1 = a_{(0\dots01)}$...and $a_{2^N-1} = a_{(1\dots11)}$. In order to satisfy this condition in Equation 3, the complex-valued amplitudes a_n are rescaled such that $\langle\Psi|\Psi\rangle = 1$. State representation of qubits in Psitrum follows the convention in quantum computing literature where the qubits are labelled from 0 to $N - 1$, that is the rightmost (leftmost) bit corresponds to the 0 ($N - 1$) qubit [60].

3.2 | Framework of psitrum

Quantum algorithms can be fed into Psitrum as string matrices in form of quantum circuits. Rows of the matrices represent N qubits, and columns represent the execution of M operations of the input algorithms. Each element of the circuit represent a gate that applies a specific unitary operation. The definition of each level of a circuit implemented in Psitrum is shown in Figure 1. This framework allows the user to initialise qubits, set parameters of quantum gates and select the output qubits to be measured.

Psitrum is a gate-model QC simulator that follows the workflow of many full-stack quantum simulators [61–64]. First, the problem is defined at a high-level of abstraction, and based on the type of the problem a quantum algorithm is selected in such a way that maximise the output probabilities of the solution. Next, the quantum algorithm is expressed in form of a quantum circuit that applies unitary operations. This circuits then needs to be compiled to a specific set of quantum gates. Finally, the circuit is executed on Psitrum which in turn acts like a quantum compiler. Figure 2 shows the workflow of Psitrum with examples and details at each level.

3.3 | Decoherence and quantum noise

Simulating quantum errors in Psitrum is possible by simply adding the unitary operation corresponding to a specific noise model. This can be done in the backend of Psitrum. Here, we show how to do that by implementing depolarising channel which is an important type of incoherent noise. The depolarising noise model is given by the following [65]:

$$\xi(\rho) = (1 - p)\rho + \frac{pI}{2^n} \quad (4)$$

where ξ denotes the depolarising noise channel, ρ is the density matrix, n is the number of qubits and p is the error rate in range $0 \leq p \leq \frac{4^n}{4^n - 1}$. The parameter p represents the probability that the qubits are depolarised and replaced with completely mixed state $\frac{I}{2^n}$, while $1 - p$ is the probability that the qubits are

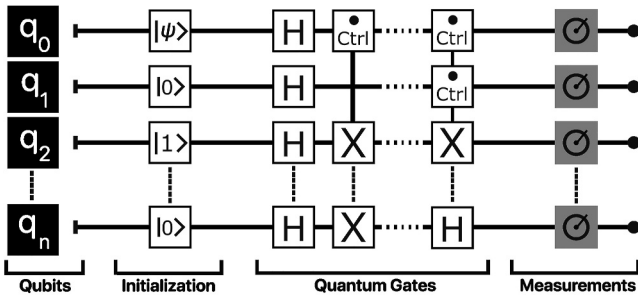


FIGURE 1 Different level of a basic quantum circuit implemented in Psitrum. This framework allows to set/reset qubits, set parameters of quantum gates and choose which output qubits to be measured.

still in their pure states ρ . Psitrum gives the choice to select p as a stochastic error rate or at overshoot. The effect of the depolarising channel is basically a uniform contraction of all axes on the Bloch sphere as function of p .

4 | USER INTERFACE AND VISUALISATION

There are few user friendly and easy-accessible software that allow to graphically design, run, and test quantum algorithms as most of the available QC simulators are SDKs or frameworks. Also, there are even fewer software that allow to represent the simulation results with variety of visualisation tools. Psitrum is a software that provides all of these services in an integrated environment. Psitrum provides a simple, friendly, and dynamic GUI, which is divided into four sections that provide the main services of the software, including circuit designer, quantum state tracer, visualisation graphs and numerical representations.

4.1 | Circuit designer

This section allows the user to graphically design quantum circuits with a defined universal set of quantum gates. The user can add qubits and stages to the circuit as many as needed, and limited only by the available computational power on the user's hardware. The user can then set the initial state of each qubit and which of the output qubits to be measured. Also, it is possible to add additional parameters to some quantum gates, like phase angles of rotations on Bloch sphere. Finally, when the user has designed circuit, next it can be executed on the

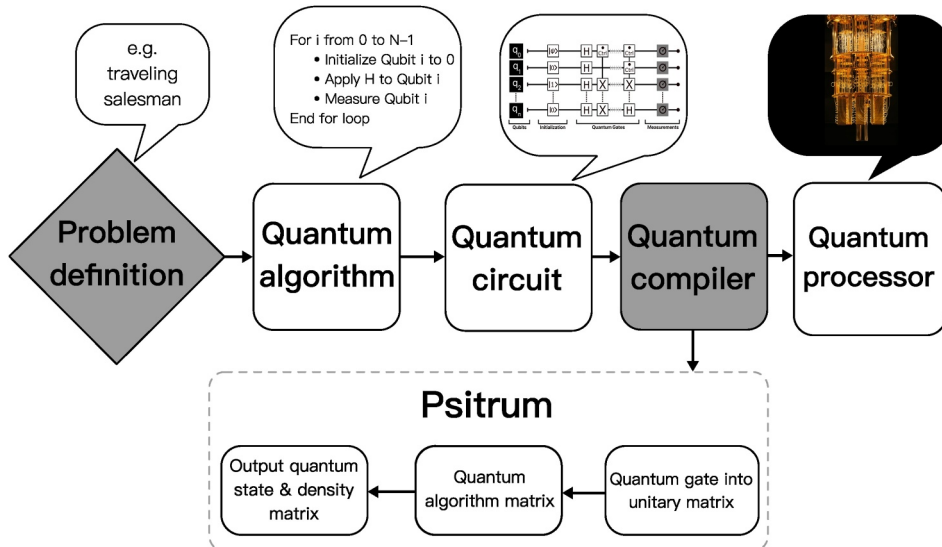


FIGURE 2 Workflow of Psitrum, which follows a full-stack gate-model quantum computer simulator [64]. In this way, the selected quantum algorithm for a specific problem needs to be expressed in form of a circuit using a set of defined gates. Psitrum then simulates the operation of an actual quantum compiler that executes this circuit. Finally, a set of visualisation tools is available for the user, including Bloch sphere diagrams, state and density tracers, 3D and heatmap diagrams.

simulator which dynamically presents the outputs in multiple result windows.

4.2 | Visualisation diagrams

Quantum computers are probabilistic systems and the output of a quantum circuit is based on probabilities. Diagrams can be useful to visualise probabilities, heatmap and Bloch Spheres. These diagrams are provided in Psitrum to visualise algorithm matrix, output states and density matrices.

4.3 | State and density tracers

The implementation of the quantum state and density matrix tracers in the GUI are another useful features provided by Psitrum. These allow the user to trace out the quantum state of all qubit and density matrix at each stage after any quantum operation. Psitrum visualises each step on N qubit in N different Bloch Sphere diagrams to show quantum states at every stage of the circuit. Each qubit is visualised by calculating its Bloch vector. The user can trace out the evolution of all qubits after each stage, either by choosing to access the Bloch vector of each stage or the output Bloch vectors after executing all operations. Similarly, Psitrum shows all density matrices after M stages with and without quantum noise.

4.4 | Numerical tables

It is useful, especially for small circuits, to track the state of qubits during the simulation. The ability to observe the numeric values of the output is useful for large circuits where visualisation tools are now more needed. In addition, Psitrum also provides several output numerical tables that can be exported, including the output table of the algorithm matrix, its density matrix and the output quantum states. The user can then save the data and have a complete reference of the solution to the simulated quantum algorithm.

5 | TESTING AND VALIDATION

This section is about testing Psitrum to validate its performance by implementing four quantum algorithms, quantum full-adder [66], Deutsch–Jozsa [67], Grover Search [68] and prime factorisation [69] algorithms. These circuits are good benchmark problems for a universal QC simulator [70–73].

5.1 | Quantum full-adder

Full-Adder circuit adds the input bits of A and B plus a carry input bit C_{in} to produce the sum S and a carry output C_{out} bits. Classical full-adder requires three input and two output bits. However, the quantum version requires the same number of

input and output qubits, since the circuit must be reversible. The truth table of the full-adder with assigned qubits is given in Table 2, and the corresponding quantum circuit is shown in Figure 3.

Next, quantum full-adder is modelled in Psitrum while introducing depolarising noise channels, given in Equation (4). Figure 4a shows a heatmap of the simulated circuit and Figure 4b shows the corresponding probabilities of the measured output qubits, matching the truth table in Table 1. Depolarising channels are introduced at all stages of the circuit in Figure 3, with $p = 0.05$ at overshoot. The effect of this noise model can be seen on the output density matrix. Figure 4c, d shows the output density matrices with and without noise. Clearly, depolarising channels reduce the amplitudes of the density matrix without introducing dephasing on qubits.

5.2 | Deutsch–Jozsa

Deutsch–Jozsa (DJ) algorithm finds whether an oracle is constant or balanced. If all outputs qubits are only zeros or ones then the function is constant, whereas if the function is balanced then exactly half the output qubits are measured to be

TABLE 2 Truth table of a full-adder circuit. A quantum full-adder requires at least five qubits with the same number of inputs and outputs so it becomes reversible. q_0 and q_1 represent the input bits with q_3 representing the input carry bit. The output sum and carry bits are represented by q_5 and q_4 , respectively.

A	B	C_{in}	S	C_{out}
q_0	q_1	q_2	q_3	q_4
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

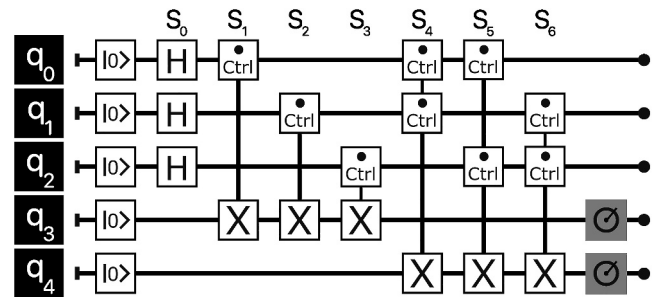


FIGURE 3 Circuit diagram of a five qubits quantum full-adder in Psitrum. The first three qubits take the input and output bits are measured in the last two qubits.

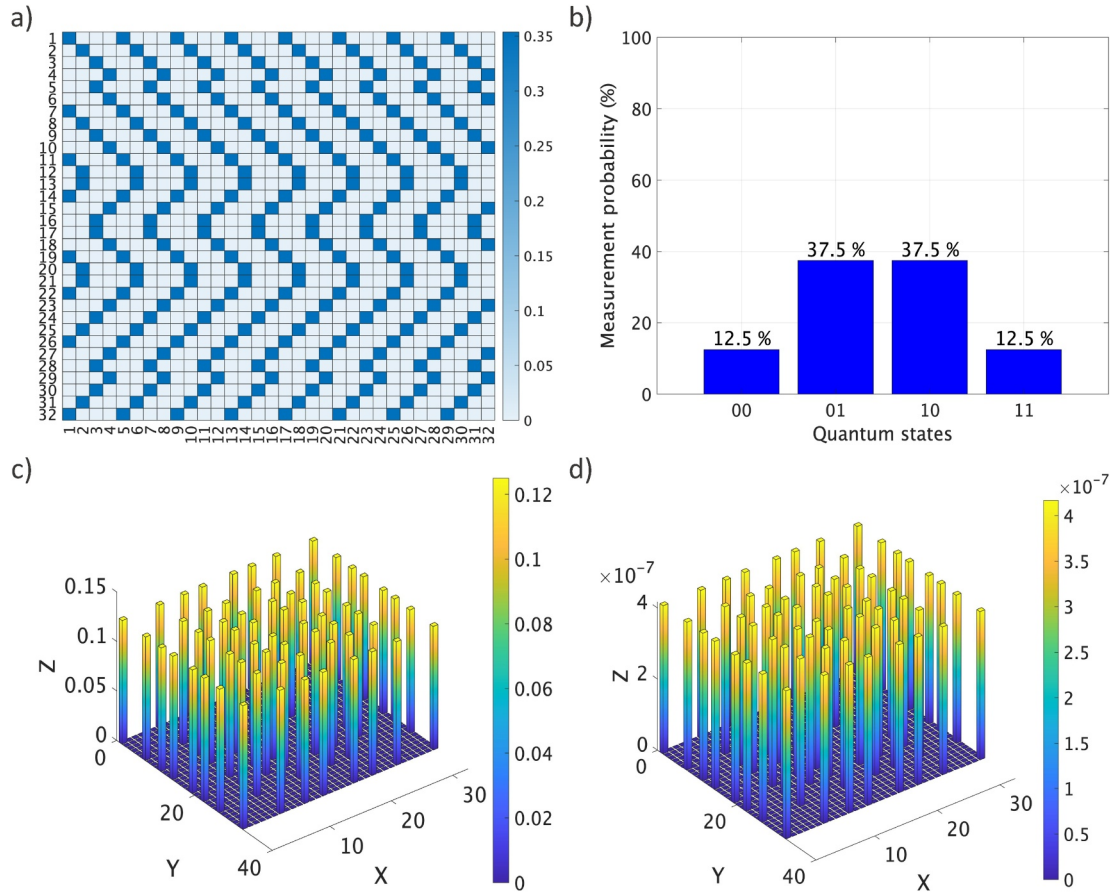


FIGURE 4 Results after running quantum full-adder in Psitrum. (a) Shows heatmap of the simulated circuit matrix, and (b) shows the output states probabilities of measured qubits. (c) and (d) show the density matrix of the output with and without depolarising noise channels, respectively. All stages are noisy depolarised with $p = 0.05$ at overshoot.

zeros and the other half are ones. To apply DJ algorithm, first all qubits are initialised to zero states followed by Hadamard gates to create superposition. Next, the circuit of the oracle to be tested is constructed followed Hadamard gates on all qubits. Finally, the upper $N-1$ qubits are measured to find out whether this function constant or balanced. Figure 5 shows DJ algorithm for a balanced function.

Next, DJ circuit is modelled in Psitrum with depolarising noise channels. Figure 6a shows a heatmap of the simulated circuit and Figure 6b shows the measured probabilities of the first four qubits. All qubits are measured at the $|1111\rangle$ state, indicating that this is a balanced function. Depolarising channels are introduced at all stages of the circuit in Figure 5, with $p = 0.05$ at overshoot. The effect of this noise model can be seen on the output density matrix. Figure 6c,d show the output density matrices with and without noise. At both cases, the measured qubits are all still at the $|1111\rangle$ state with much smaller amplitudes.

5.3 | Grover search

Grover algorithm requires iterations that scale as the square root of length of the list. This is a quadratic speed over classical

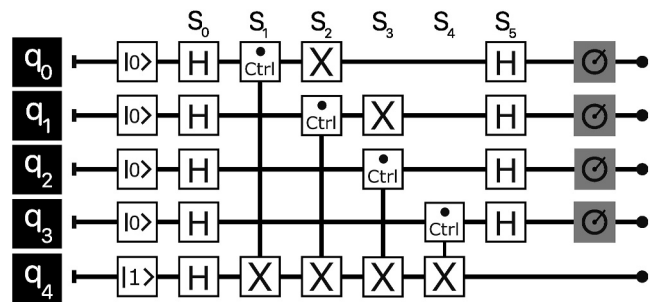


FIGURE 5 Diagram of a Deutsch-Jozsa circuit of a balanced function in Psitrum. The first four qubits are measured to find out whether the oracle is balanced or constant.

algorithms. First, qubits are initialised at the desired state to be found, followed by Hadamard gates to create superposition states. Next, Grover circuit applies selective phase inversion of the states followed by inversion about the mean in order to amplify the probability of measuring the correct state. Figure 7 shows two iterations of Grover algorithm for three-qubit search of $|110\rangle$ state.

Next, Grover circuit is modelled in Psitrum with depolarising noise channels. Figure 8a shows a heatmap of the

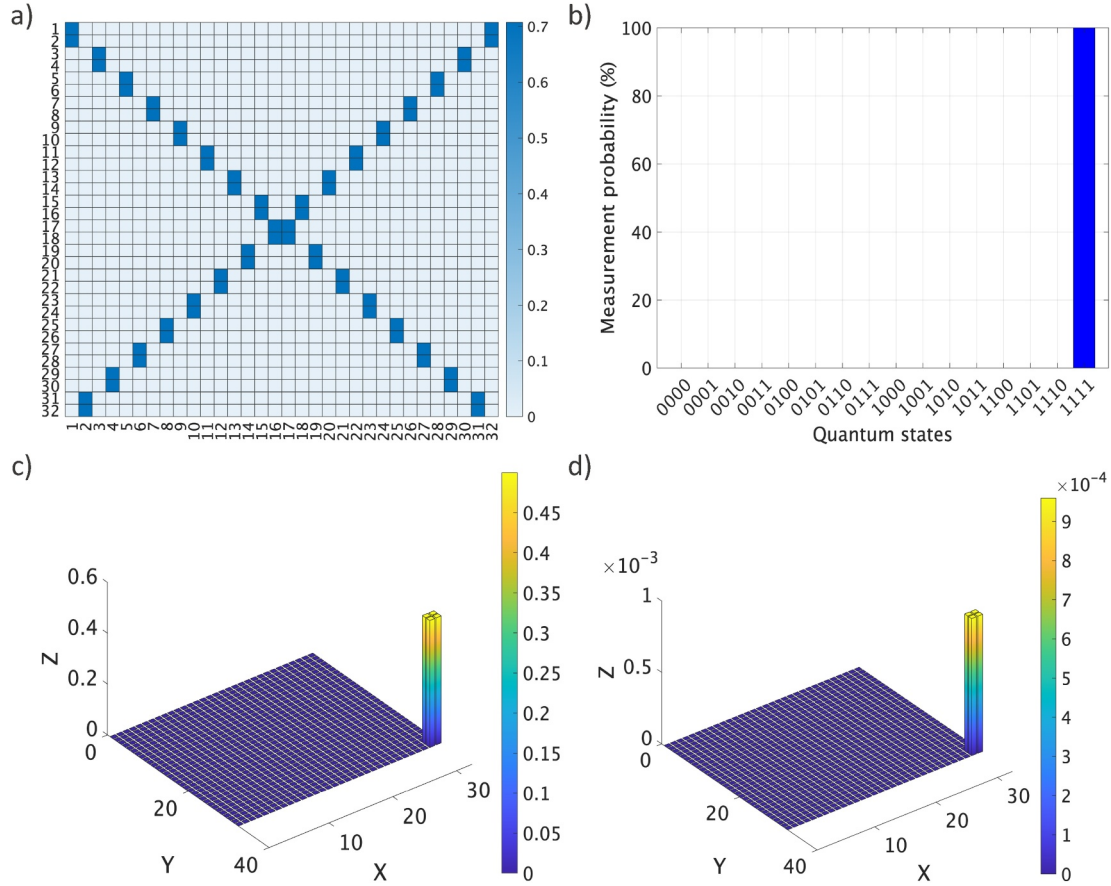


FIGURE 6 Output of running Deutsch–Jozsa algorithm of a balanced function. (a) Shows heatmap of the simulated circuit matrix, and (b) shows the output states probabilities of measured qubits. (c) and (d) show the density matrix of the output with and without depolarising noise channels, respectively. All stages are noisy depolarised with $p = 0.05$ at overshoot.

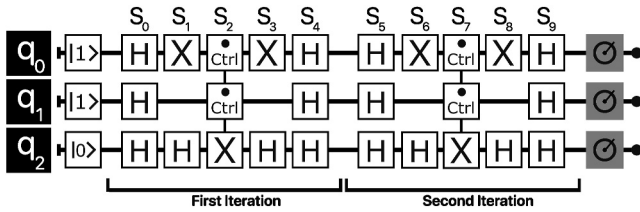


FIGURE 7 Circuit diagram of two iterations of three qubits search algorithm. The qubit states are initialised at $|110\rangle$ representing the state to be searched.

simulated circuit and Figure 8b shows the measured probabilities of qubits after the second iteration. The correct answer is found in just two iterations, in which standard classical search would require at least eight iterations. Depolarising channels are introduced at all stages of the circuit in Figure 7, with $p = 0.05$ at overshoot. The effect of this noise model can be seen on the output density matrix. Figure 8c,d show the output density matrices with and without noise. The correct state still appears after depolarising channels with smaller amplitudes.

5.4 | Prime factorisation using variational quantum eigensolver

Unlike Shors algorithm that makes use of the period to compute the factors of the number to be factorised [73], here we compute factors of a given number through solving an optimisation problem. The cost function used is given by $(N - pq)^2$, where N is the number to be factorised, while p and q are factors to be identified and expressed in binary form as qubits over which the optimisation is performed. Refer [69] for a complete discussion on how the cost function is constructed and its complexity. The authors there made use of imaginary time evolution to solve for the factors. Here, instead we only make use of the gradients to the cost function through standard Variational Quantum EigenSolver (VQE). We would like to note that the updates in the imaginary time evolution differs from VQE only by a factor of the fisher information that works similar to the hessian for gradient updates.

We use a hardware efficient ansatz (Figure 9) to implement a variational circuit that consists of repeating layers to allow for a low error in the cost function output, but not representative enough to result in barren plateaus. The initial parameters of

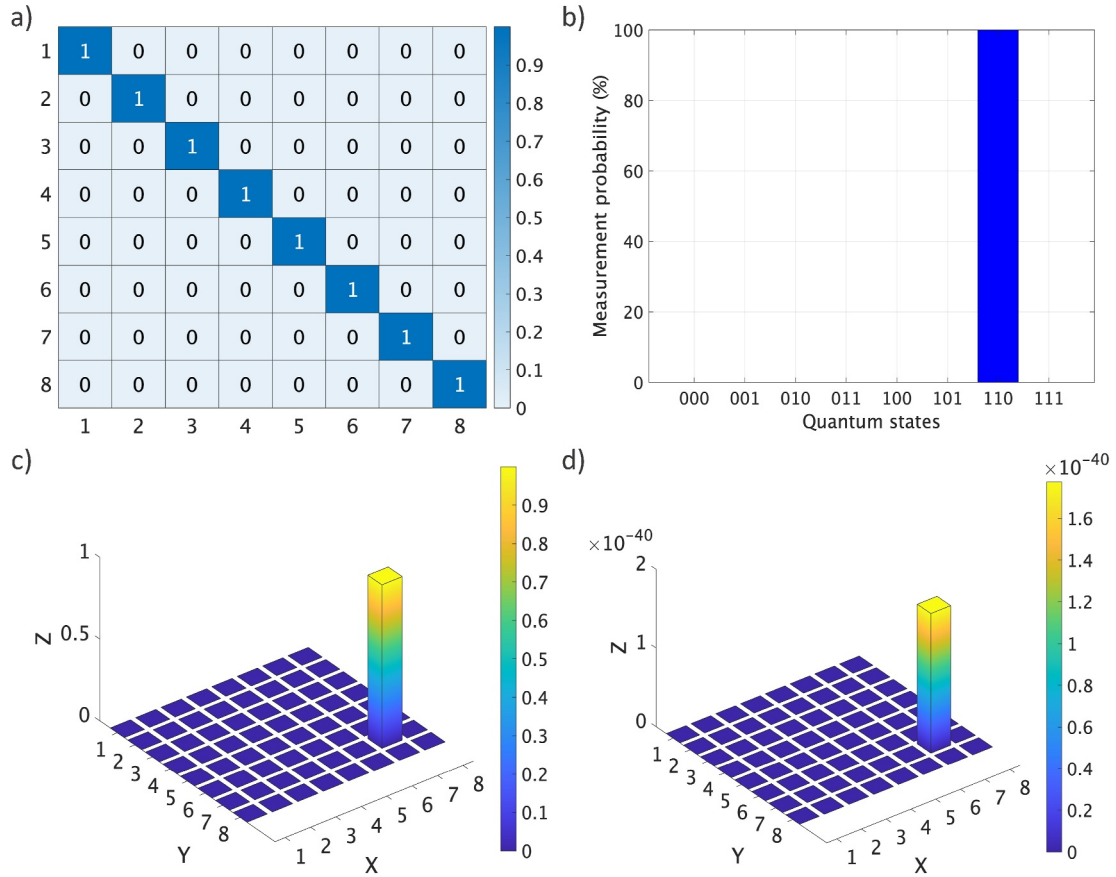


FIGURE 8 Results of three qubits Grover algorithm. (a) Shows heatmap of the simulated circuit matrix, and (b) shows the output states probabilities of measured qubits after the second iteration. (c) and (d) show the density matrix of the output with and without depolarising noise channels, respectively. All stages are noisy depolarised with $p = 0.05$ at overshoot.

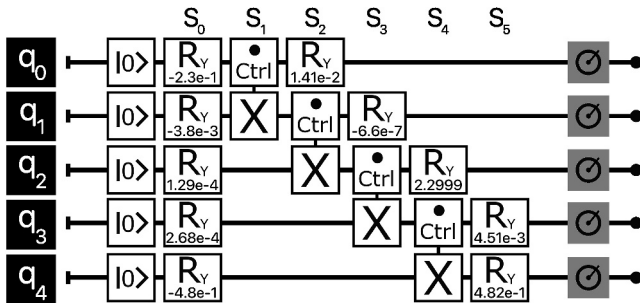


FIGURE 9 Diagram of five-qubit variational circuit used to factorise 91. Parameters of the circuits were randomly initialised.

the circuit were randomly initialised. The learning rate was set to 0.1 and the convergence threshold for the amplitude of factors was set to 0.90 at least. Results of simulating this variational circuit for factorising 91 are shown in Figure 10.

We plot training of the cost function and the amplitude of the solutions at the end of 100 iterations for the factorisation of numbers 77 and 91 starting with the same set of initial parameters in either case and compare the results with IBM Qiskit Aqua framework (Figure 11). 77 has factors of 11 (1101) and 7 (111), and 91 has factors of 13 (1110) and 7 (111). Given

that the least significant digit of all prime numbers begin with 1, we have a total of five qubits over which the optimisation is to be performed.

6 | CONCLUSION

We presented Psitrum, a universal QC simulator based on classical hardware and showed how to run widely popular quantum algorithms, namely quantum-full adder, Deutsch–Jozsa and Grover search, both in the presence and absence of quantum gate noise. We made use of visualisation tools available in the software to demonstrate the simulation of quantum circuits pertaining to these algorithms on Psitrum. In addition, we solved the factorisation problem using standard VQE and were able to run the circuit in Psitrum to factorise prime numbers. This demonstrates that, Psitrum provides features for simulating noisy and noiseless quantum circuits to solve a wide class of quantum algorithms available for NISQ processors. Given that Psitrum runs on a local server with a simplistic MATLAB interface, it provides a base layer for developers to easily add their own customisation to suit their needs. In future, we are going to add modules relevant to implement various machine learning methods directly onto this

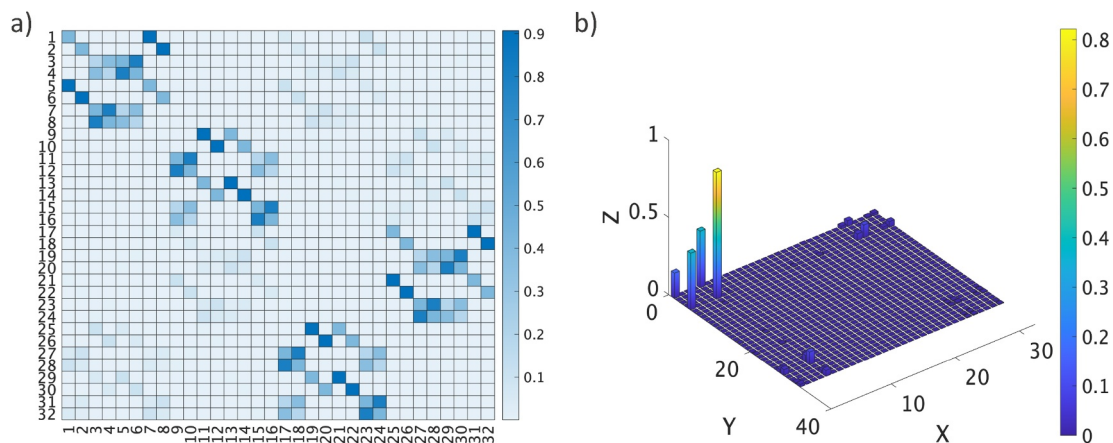


FIGURE 10 Output of simulating the variational circuit for factoring 91 in Psitrum. (a) Shows heatmap of the circuit matrix, and (b) shows the density matrix of the output state.

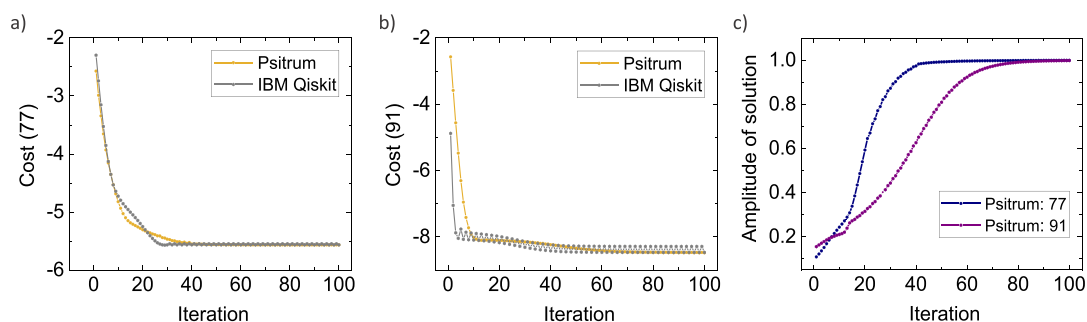


FIGURE 11 Five-qubit factorisation examples. (a) and (b) show the minimisation of the cost function for 77 and 91, respectively, using Psitrum and IBM Qiskit [27]. (c) Shows the amplitude of the solutions in the computational basis for factoring 77 and 91, respectively, as function of iteration solved by Psitrum.

platform, which will help to provide a good starting point to newcomers for a good experience with our visual learning tools.

AUTHOR CONTRIBUTIONS

Mohammed Alghadeer: Conceptualisation; data curation; formal analysis; investigation; methodology; project administration; resources; software; validation; visualisation; writing – original draft; writing – review & editing. **Eid Aldawsari:** Data curation; methodology; software; validation; visualisation. **Raja Selvarajan:** Validation; writing – review & editing. **Khaled Alutaibi:** Supervision; writing – review & editing. **Sabre Kais:** Supervision; writing – original draft. **Fahhad H. Alharbi:** Resources; supervision; writing – original draft; writing – review & editing.

ACKNOWLEDGEMENTS

This work was supported by the Deanship of Scientific Research at King Fahd University of Petroleum and Minerals. S.K. acknowledge the support from the National Science Foundation under award number 1955907.

CONFLICT OF INTEREST STATEMENT

The authors declare no competing interests that could influence the research outcomes or the presentation of results.

DATA AVAILABILITY STATEMENT

The datasets generated during the current study are available upon reasonable request from the corresponding author.

CONSENT FOR PUBLICATION

Participants provided explicit consent for the publication of research findings, ensuring the privacy and confidentiality of their personal information.

ORCID

Mohammed Alghadeer  <https://orcid.org/0000-0003-1763-0757>

REFERENCES

1. Steane, A.: Quantum computing. *Rep. Prog. Phys.* 61(2), 117–173 (1998). <https://doi.org/10.1088/0034-4885/61/2/002>
2. Ryan, C., et al.: Liquid-state nuclear magnetic resonance as a testbed for developing quantum control methods. *Phys. Rev.* 78(1), 012328 (2008). <https://doi.org/10.1103/physreva.78.012328>
3. Berry, D.W., et al.: Efficient quantum algorithms for simulating sparse Hamiltonians. *Commun. Math. Phys.* 270(2), 359–371 (2007). <https://doi.org/10.1007/s00220-006-0150-x>
4. Benenti, G., et al.: Efficient quantum computing of complex dynamics. *Phys. Rev. Lett.* 87(22), 227901 (2001). <https://doi.org/10.1103/physrevlett.87.227901>
5. Buluta, I., Nori, F.: Quantum simulators. *Science* 326(5949), 108–111 (2009). <https://doi.org/10.1126/science.1177838>

6. Brylinski, J.-L., Brylinski, R.: Universal quantum gates. In: *Mathematics of Quantum Computation*, pp. 117–134. Chapman and Hall/CRC (2002)
7. Roushan, P., et al.: Observation of topological transitions in interacting quantum circuits. *Nature* 515(7526), 241–244 (2014). <https://doi.org/10.1038/nature13891>
8. Aspuru-Guzik, A., Walther, P.: Photonic quantum simulators. *Nat. Phys.* 8(4), 285–291 (2012). <https://doi.org/10.1038/nphys2253>
9. Manin, Y.: *Computable and Uncomputable*, pp. 128. Sovetskoye Radio, Moscow (1980)
10. Feynman, R.P.: Simulating physics with computers. In: *Feynman and Computation*, pp. 133–153. CRC Press (2018)
11. Nakahara, M.: *Quantum Computing: From Linear Algebra to Physical Realizations*. CRC press (2008)
12. Li, S.-S., et al.: Quantum computing. *Proc. Natl. Acad. Sci. USA* 98(21), 11847–11848 (2001). <https://doi.org/10.1073/pnas.191373698>
13. Britton, J.W., et al.: Engineered two-dimensional ising interactions in a trapped-ion quantum simulator with hundreds of spins. *Nature* 484(7395), 489–492 (2012). <https://doi.org/10.1038/nature10981>
14. Jones, N.C., et al.: Layered architecture for quantum computing. *Phys. Rev. X* 2(3), 031007 (2012). <https://doi.org/10.1103/physrevx.2.031007>
15. Leuenberger, M.N., Loss, D.: Quantum computing in molecular magnets. *Nature* 410(6830), 789–793 (2001). <https://doi.org/10.1038/35071024>
16. De Raedt, K., et al.: Massively parallel quantum computer simulator. *Comput. Phys. Commun.* 176(2), 121–136 (2007). <https://doi.org/10.1016/j.cpc.2006.08.007>
17. Obenland, K.M., Despain, A.M.: A parallel quantum computer simulator. arXiv preprint quant-ph/9804039 (1998)
18. Deutsch, D.: Quantum theory, the church–turing principle and the universal quantum computer. *Proc. R. Soc. Lond. A. Math. Phys. Sci.* 400(1818), 97–117 (1985)
19. Fortnow, L.: One complexity theorist’s view of quantum computing. *Electron. Notes Theor. Comput. Sci.* 31, 58–72 (2000). [https://doi.org/10.1016/s1571-0661\(05\)80330-5](https://doi.org/10.1016/s1571-0661(05)80330-5)
20. Preskill, J.: Quantum computing in the nisq era and beyond. *Quantum* 2, 79 (2018). <https://doi.org/10.22331/q-2018-08-06-79>
21. Murali, P., et al.: Software mitigation of crosstalk on noisy intermediate-scale quantum computers. In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1001–1016 (2020)
22. Karafyllidis, I.G.: Quantum computer simulator based on the circuit model of quantum computation. *IEEE Trans. Circuits Syst. Regular Pap.* 52(8), 1590–1596 (2005). <https://doi.org/10.1109/tcsi.2005.851999>
23. Khammassi, N., et al.: Qx: a high-performance quantum computer simulation platform. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 464–469. IEEE (2017)
24. Guzik, V., et al.: Models of a quantum computer, their characteristics and analysis. In: *2015 9th International Conference on Application of Information and Communication Technologies (AICT)*, pp. 583–587. IEEE (2015)
25. Arute, F., et al.: Quantum supremacy using a programmable superconducting processor. *Nature* 574(7779), 505–510 (2019). <https://doi.org/10.1038/s41586-019-1666-5>
26. Ball, P.: Google moves closer to a universal quantum computer. *Nat. News* (2016). <https://doi.org/10.1038/nature.2016.20032>
27. Cross, A.: The IBM Q experience and qiskit open-source quantum computing software. *APS March Meet. Abstr.* 2018, L58003 (2018)
28. Hastings, M.B., et al.: Improving quantum algorithms for quantum chemistry. arXiv preprint arXiv:1403.1539 (2014)
29. Steiger, D.S., Häner, T., Troyer, M.: Projectq: an open source software framework for quantum computing. *Quantum* 2, 49 (2018). <https://doi.org/10.22331/q-2018-01-31-49>
30. Liu, W., et al.: An optimized quantum minimum searching algorithm with sure-success probability and its experiment simulation with cirq. *J. Ambient Intell. Hum. Comput.* 12(11), 1–10 (2021). <https://doi.org/10.1007/s12652-020-02840-z>
31. Budhrani, R.: *Quantumsim: a memory efficient simulator for quantum computing* (2020)
32. Guerreschi, G.G., et al.: Intel quantum simulator: a cloud-ready high-performance simulator of quantum circuits. *Quantum Sci. Technol.* 5(3), 034007 (2020). <https://doi.org/10.1088/2058-9565/ab8505>
33. Jones, T., et al.: Qwest and high performance simulation of quantum computers. *Sci. Rep.* 9(1), 1–11 (2019). <https://doi.org/10.1038/s41598-019-47174-9>
34. Wu, S.L., et al.: Application of quantum machine learning using the quantum variational classifier method to high energy physics analysis at the lhc on ibm quantum computer simulator and hardware with 10 qubits. *J. Phys. G Nucl. Part. Phys.* 48(12), 125003 (2021). <https://doi.org/10.1088/1361-6471/ac1391>
35. Gidney, C.: Quirk quantum circuit simulator, A drag-and-drop quantum circuit simulator. URL: <https://algassert.com/quirk> (2016)
36. Zhou, Y., Stoudenmire, E.M., Waintal, X.: What limits the simulation of quantum computers? *Phys. Rev. X* 10(4), 041038 (2020). <https://doi.org/10.1103/physrevx.10.041038>
37. Zalka, C.: Efficient simulation of quantum systems by quantum computers. *Fortschritte der Physik: Progress of Physics. Fortschr. Phys.* 46(6–8), 877–879 (1998). [https://doi.org/10.1002/\(sic\)1521-3978\(199811\)46:6/8<877::aid-prop877>3.0.co;2-a](https://doi.org/10.1002/(sic)1521-3978(199811)46:6/8<877::aid-prop877>3.0.co;2-a)
38. Verstraete, F., et al.: Four qubits can be entangled in nine different ways. *Phys. Rev.* 65(5), 052112 (2002). <https://doi.org/10.1103/physreva.65.052112>
39. Avron, J.E., Bisker, G., Kenneth, O.: Visualizing two qubits. *J. Math. Phys.* 48(10), 102107 (2007). <https://doi.org/10.1063/1.2795217>
40. Yepez, J.: Relativistic path integral as a lattice-based quantum algorithm. *Quant. Inf. Process.* 4(6), 471–509 (2005). <https://doi.org/10.1007/s11128-005-0009-7>
41. Huggins, W., et al.: Towards quantum machine learning with tensor networks. *Quant. Sci. Technol.* 4(2), 024001 (2019). <https://doi.org/10.1088/2058-9565/aaea94>
42. Maciejewski, F.B., Zimborás, Z., Oszmaniec, M.: Mitigation of readout noise in near-term quantum devices by classical post-processing based on detector tomography. *Quantum* 4, 257 (2020). <https://doi.org/10.22331/q-2020-04-24-257>
43. Nachman, B., et al.: Unfolding quantum computer readout noise. *npj Quant. Inf.* 6(1), 1–7 (2020). <https://doi.org/10.1038/s41534-020-00309-7>
44. Gutiérrez, M., et al.: Errors and pseudothresholds for incoherent and coherent noise. *Phys. Rev.* 94(4), 042338 (2016). <https://doi.org/10.1103/physreva.94.042338>
45. Wood, C.J., Gambetta, J.M.: Quantification and characterization of leakage errors. *Phys. Rev.* 97(3), 032306 (2018). <https://doi.org/10.1103/physreva.97.032306>
46. Urbanek, M., et al.: Mitigating depolarizing noise on quantum computers with noise-estimation circuits. *Phys. Rev. Lett.* 127(27), 270502 (2021). arXiv preprint arXiv:2103.08591. <https://doi.org/10.1103/physrevlett.127.270502>
47. Wallman, J.J., Emerson, J.: Noise tailoring for scalable quantum computation via randomized compiling. *Phys. Rev.* 94(5), 052325 (2016). <https://doi.org/10.1103/physreva.94.052325>
48. Cai, Z., Xu, X., Benjamin, S.C.: Mitigating coherent noise using pauli conjugation. *npj Quant. Inf.* 6(1), 1–9 (2020). <https://doi.org/10.1038/s41534-019-0233-0>
49. MathWorks, Mathworks: company overview. <https://ch.mathworks.com/content/dam/mathworks/handout/2022-company-factsheet-8-5x11-8282v22.pdf> (2022). Accessed 08 Aug 2023
50. MathWorks: Teach and learn with matlab and simulink. https://www.mathworks.com/academia.html?tid=gn_acad (2022). Accessed 08 Aug 2023
51. Singh, C.: Student understanding of quantum mechanics. *Am. J. Phys.* 69(8), 885–895 (2001). <https://doi.org/10.1119/1.1365404>
52. Müller, R., Wiesner, H.: Teaching quantum mechanics on an introductory level. *Am. J. Phys.* 70(3), 200–209 (2002). <https://doi.org/10.1119/1.1435346>
53. Koupilová, Z., Káčovský, P.: *Interactive Applets in Introductory Course of Quantum Physics: Their Role Not Only in Distance Learning*, vol. 2458. AIP Publishing (2022). <https://doi.org/10.1063/5.0078619.030017>

54. Ahmed, S.Z., et al.: Student use of a quantum simulation and visualization tool. *Eur. J. Phys.* 43(6), 065703 (2022). <https://doi.org/10.1088/1361-6404/ac93c7>
55. Waitzmann, M., et al.: Key experiment and quantum reasoning. *Physics* 4(4), 1202–1229 (2022). <https://doi.org/10.3390/physics4040078>
56. Hu, P., Li, Y., Singh, C.: Challenges in addressing student difficulties with time-development of two-state quantum systems using a multiple-choice question sequence in virtual and in-person classes. *Eur. J. Phys.* 43(2), 025704 (2022). <https://doi.org/10.1088/1361-6404/ac49f4>
57. Marshman, E., Singh, C.: Quilts: validated teaching–learning sequences for helping students learn quantum mechanics. In: *Physics Teacher Education: What Matters?* pp. 15–35. Springer (2022)
58. Gottesman, D., Chuang, I.L.: Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature* 402(6760), 390–393 (1999). <https://doi.org/10.1038/46503>
59. Gilbert, J.R., Moler, C., Schreiber, R.: Sparse matrices in matlab: design and implementation. *SIAM J. Matrix Anal. Appl.* 13(1), 333–356 (1992). <https://doi.org/10.1137/0613024>
60. Nielsen, M.A., Chuang, I.: *Quantum computation and quantum information* (2002)
61. Bertels, K., et al.: Quantum computer architecture: towards full-stack quantum accelerators. In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, pp. 1–6 (2020)
62. Frey, V., et al.: Programming the full stack of an open-access quantum computer. *arXiv preprint arXiv:2106.06549* (2021)
63. Bassman, L., Powers, C., de Jong, W.A.: Arqtic: a full-stack software package for simulating materials on quantum computers. *arXiv preprint arXiv:2106.04749* (2021)
64. Fingerhuth, M., Babej, T., Wittek, P.: Open source software in quantum computing. *PLoS One* 13(12), e0208561 (2018). <https://doi.org/10.1371/journal.pone.0208561>
65. Nielsen, M.A., Chuang, I.L.: Quantum computation and quantum information. *Phys. Today* 54(2), 60–62 (2001). <https://doi.org/10.1063/1.1428442>
66. Cheng, K.-W., Tseng, C.-C.: Quantum full adder and subtractor. *Electron. Lett.* 38(22), 1343–1344 (2002). <https://doi.org/10.1049/el:20020949>
67. Deutsch, D., Jozsa, R.: Rapid solution of problems by quantum computation. *Proc. R. Soc. Lond. Ser. A: Math. Phys. Sci.* 439(1907), 553–558 (1992)
68. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pp. 212–219 (1996)
69. Selvarajan, R., et al.: Prime factorization using quantum variational imaginary time evolution. *Sci. Rep.* 11(1), 1–8 (2021). <https://doi.org/10.1038/s41598-021-00339-x>
70. Seyedi, S., Navimipour, N.J.: An optimized design of full adder based on nanoscale quantum-dot cellular automata. *Optik* 158, 243–256 (2018). <https://doi.org/10.1016/j.ijleo.2017.12.062>
71. Gulde, S., et al.: Implementation of the deutsch–jozsa algorithm on an ion-trap quantum computer. *Nature* 421(6918), 48–50 (2003). <https://doi.org/10.1038/nature01336>
72. Jones, J.A., Mosca, M., Hansen, R.H.: Implementation of a quantum search algorithm on a quantum computer. *Nature* 393(6683), 344–346 (1998). <https://doi.org/10.1038/30687>
73. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* 41(2), 303–332 (1999). <https://doi.org/10.1137/s0036144598347011>

How to cite this article: Alghadeer, M., et al.: Psitrum: an open source simulator for universal quantum computers. *IET Quant. Comm.* 1–15 (2024). <https://doi.org/10.1049/qt2.12101>

APPENDIX A: USER INTERFACE IN PSITRUM

Psitrum software and source codes are available GitHub with an installation guide that includes all the necessary details. The GUI in Psitrum is structured to provide seamless navigation and interaction with quantum circuits. A clean and minimalistic design philosophy ensures that users can focus on the task at hand without distraction.

Psitrum allows users to design quantum circuits using Psitrum's unique functionality that allows users to quickly select from a comprehensive set of quantum gates and combine them to construct complex algorithms. In addition, a range of visualisation tools, including state vectors, Bloch spheres, and probability distributions, are available to interpret the results of quantum simulations (Figure A1).



FIGURE A1 The Psitrum user interface is engineered to facilitate efficient navigation and interaction within the quantum circuit framework. It adheres to a minimalist design aspects, optimising the user's cognitive focus on simulation tasks by reducing visual clutter and interface complexities. The example shown here is for circuit of a five qubits quantum to simulate a full-adder using the GUI of Psitrum. The first three qubits take the input and output bits are measured in the last two qubits (check the results for more details).

APPENDIX B: QUANTUM GATES IN PSITRUM

This section provides details of gate operations implemented in Psitrum. These are the basic quantum gates for many algo-

rithms and are available in this first version of Psitrum. In later versions, we can easily add more gates. Definitions of single- and multi-qubits gates are given in Table B1.

TABLE B1 Single- and multi-qubits quantum gates defined in Psitrum.




















Gate	Matrix	Symbol
I	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	
X	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	
Y	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	
Z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	
H	$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	
S	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$	
T	$\begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix}$	
S [†]	$\begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}$	
T [†]	$\begin{bmatrix} 1 & 0 \\ 0 & e^{-\frac{i\pi}{4}} \end{bmatrix}$	
U ₁	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$	
U ₂	$\begin{bmatrix} 1 & -e^{i\theta} \\ e^{i\phi} & e^{i(\theta+\phi)} \end{bmatrix}$	
U ₃	$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda} \sin\left(\frac{\theta}{2}\right) \\ e^{i\phi} \sin\left(\frac{\theta}{2}\right) & e^{i(\lambda+\theta)} \cos\left(\frac{\theta}{2}\right) \end{bmatrix}$	

TABLE B1 (Continued)

Gate	Matrix	Symbol
R_X	$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}$	
R_Y	$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}$	
R_Z	$\begin{bmatrix} e^{-\frac{i\theta}{2}} & 0 \\ 0 & e^{\frac{i\theta}{2}} \end{bmatrix}$	
S_X	$\begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix}$	
S_X^\dagger	$\begin{bmatrix} 1-i & 1+i \\ 1+i & 1-i \end{bmatrix}$	
SWAP	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	
CNOT	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	
Toffoli	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$	