Predicting the fidelity of Quantum Circuits

Search for better metrics for the Qubit Routing Problem

Diogo Manuel Antunes Lopes Valada

Thesis to obtain the Master of Science Degree in

Engineering Physics

Supervisors: Prof. João Carlos Carvalho de Sá Seixas
Dr. Carmen García Almudever

Examination Committee

Chairperson: Prof. Pedro Miguel Félix Brogueira
Supervisor: Prof. João Carlos Carvalho de Sá Seixas
Members of the Committee: Prof. Yasser Rashid Revez Omar
Prof. Pedro José Gonçalves Ribeiro

October 2020

Acknowledgments

I would like to start by expressing my gratitude to Dr. Carmen Almudéver for the willingness to receive me as a Master Thesis student at her research group and the opportunity to perform this research work. Thank you for all your guidance and patience during this journey. I would also like to thank my supervisor from Instituto Superior Técnico, Dr. João Seixas, for facilitating this opportunity.

I also had the opportunity to work alongside all the other great colleagues at the Quantum Computer Engineering department, especially in the Quantum Computer Architecture Lab. They really made me feel at home and as a member of the team. I appreciate all the help provided to me by Hans van Sommeren during all the development and debugging journeys we went through together. Thank you Erik for all the support with server-related issues. And thank you Aritra Sarkar, Amitabh Yadav, Peter-Jan Derks, Abid Mouedenne, for all the interesting discussions and all the fun time together. I would also like to thank my friends from Portugal for the little taste of home through our voice calls, especially during the pandemic, and in particular Pedro Ferreira, David Neto and Luís Franco for all the useful insights.

Finally, I'd like to thank my parents, whose support made my stay in the Netherlands possible, my family, and my friend Lotte Olthof, for all the support and motivation to never stop aspiring for greater goals.

To all of you, my honest gratitude.

Diogo Valada
October 2020

Resumo

A odisseia de desenvolvimento de um computador quântico operacional é um processo multidisciplinar, que requer uma combinação de software e hardware complexa. Em particular, dado o facto que os algorítmos quanticos são desenvolvidos de modo independente em relação aos diferentes hardwares, é necessário que um compilador transforme os primeiros em códigos executáveis nos diferentes dispositivos quânticos, tendo em conta as limitações arquiteturais destes, tal como o facto que estes dispositivos quânticos raramente proporcionam conexões completa entre todos os qubits. Este processo resulta frequentemente em penalizações em termos de numero acrescido de portas quânticas e tempo de execução acrescido de circuitos quânticos. Isto é problemático, uma vez que os atuais dispositivos (NISQ, significando Noisy Intermediate-Scale Quantum), dispõem de um número reduzido de qubits, tornando técnicas de correção de erros inviáveis. Este facto cria, por sua vez, a necessidade de tornar o processo de compilação o mais eficiente possível. Neste trabalho, o problema de mapeamento de qubits, onde os qubits virtuais de algoritmos quânticos são atribuidos aos qubits físicos do dispositivo quântico, é abordado: de modo a potenciar os algoritmos de mapeamente existentes, métricas tanto novas como existentes são implementadas e testadas. Entre estas, propomos também uma métrica baseada em aprendizagem profunda capaz de superar todas as métricas anteriores para determinados tipos de algoritmos de mapeamento.

Palavras-Chave

Computação Quântica; Mapeamento de Qubits; Métricas de mapeamento; Previsão de fidelidade; Aprendizagem profunda.

Abstract

The endeavor to build a working quantum computer is highly interdisciplinary in nature, requiring a complex software-hardware stack. In particular, given the hardware-agnostic properties of common quantum algorithms, a compiler is required to transform those algorithms into code runnable by the specific quantum devices. Such compilers need to take into account the architectural constraints of the said devices, such as the fact that quantum devices do not commonly offer all-to-all connectivity. This results oftentimes on gate number and circuit depth overhead. This can become problematic, since in the current NISQ (Noisy Intermediate-Scale Quantum) era, quantum devices are characterized by high error rates and reduced number of qubits, making quantum error correction techniques unpractical: this therefore requires the compilation procedure to be as efficient as possible. In this work, the qubit mapping problem of compilation, where the software's virtual qubits are assigned to the device's physical qubits, is approached: in order to leverage the power of the existing mapping algorithms, both existing and new metrics are developed and tested in this work. Among these, we propose a deep learning-based metric that is able to outperform all the previously proposed ones for some types of mapping algorithms.

Keywords

Quantum computing; Qubit Mapping; Mapping metrics; Fidelity prediction; Deep Learning.

Contents

List of Figures

List of Tables

Acronyms

BCE	Binary Crossentropy
BN	Batch Normalization
CNOT	Controlled NOT
CPU	Central Processing Unit
CZ	Controlled Z
DAG	Directed Acyclic Graph
DL	Deep Learning
DRAM	Dynamic Random-Access Memory
EPR	Einstein, Podolsky, Rosen
ESP	Estimated Success Probability
FT	Fault Tolerance
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
GST	Gate Set Tomography
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MSE	Mean Squared Error
NISQ	Noisy Intermediate-Scale Quantum
NN	Neural Network
NNESP	Neural Network-Estimated Success Probability
NP	Nondeterministic polynomial time
QC	Quantum Computing
QEC	Quantum Error Correction

QISA	Quantum Instruction Set Architecture
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SMT	Satisfiability Modulo Theories
Tanh	Hyperbolic Tangent
TESP	Tracked Estimated Success Probability

1

Introduction

Contents

“Nature isn’t classical, dammit, and if you want to make a simulation of nature, you’d better make it quantum mechanical, and by golly it’s a wonderful problem, because it doesn’t look so easy.” These were the words of Richard Feynman, in a 1981 talk [1], where he also proposed a basic model for a quantum computer. Several milestones followed in the same decade, laying the foundations for the quantum computer.

A working quantum computer may provide very impactful capabilities, like the ability efficient simulate physical systems (think being able to simulate and develop better drugs, batteries, solar cells, among others). Additionally, it could also prove useful for applications in various other fields, such as optimization and artificial intelligence. Such a technology would also dampen the effects of Moore’s Law on the computing world, which is starting to halt the speedup of classical computers.

The performance improvement that quantum computers can offer doesn’t come from faster instruction or operations, but rather from a decrease in algorithmic complexity: this means that algorithms using the quantum information framework can perform calculations in less steps than their classical counterparts.

This makes it one of the most fascinating upcoming technologies being developed in the 21st century, along with artificial intelligence, CRISPR, clean energies (fusion energy, renewables and new generation fission reactors), distributed ledgers and space accessibility technologies, and promises to be one of the most disruptive in several areas of science, with even more applications waiting to be discovered.

Great strides have been made in developing a functional quantum computer, with several possible implementation technologies in development. However, all of the technologies still only support a small number of qubits, the basic unit of quantum information, meaning that only small programs can be executed. Furthermore, most implementations are plagued by short decoherence times (introducing low success rates for long circuits) and imperfect quantum operations (making circuits with many gates more prone to error).

The main strategy to solve the errors introduced by qubit decoherence and faulty operations is known as Quantum Error Correction (QEC) [2], which proposes encoding one logical qubit using several physical qubits. However, using several physical qubits to encode a single logical qubit means that this strategy is out of range for near future quantum computers, due to the small number of available qubits.

This fact, in turn, reduces the utility of near-term quantum computers to certain types of applications which are more vulnerable to the aforementioned problems. This class of quantum computers is commonly referred to as Noisy Intermediate-Scale Quantum (NISQ) devices, a term introduced by John Preskill in his 2019 article [3], where he also reviews the most likely applications for such devices.

1.1 Problem Definition

In order to be able to execute quantum algorithms on a given quantum device, they have to be compiled to instructions the machine supports. Different types of architectural constraints are exist, among which the Qubit Mapping is highlighted in this work. The qubit mapping problem arises to the fact that most of quantum chip implementations do not support all-to-all qubit connectivity: such connectivity is required to perform multi-qubit gates. This gives rise to the necessity of emulating the all-to-all connectivity that raw quantum algorithms typically consider in a non-all-to-all connected chip. This constitutes the essence of the Mapping problem, which is usually part of compilation step of a working quantum computer system stack.

Such emulation needs to be as efficient as possible. In other words, it needs to introduce as little overhead as possible, due to the fact that quantum computers are error-prone, and that the current NISQ quantum computers have a limited ability to correct them (due to the lack of viable Quantum Error Correction (QEC)). That is, the goal of a good mapping procedure is to make the algorithms compliant to the quantum chip's capabilities, while introducing the least amount of errors in the computation.

However, the problem itself is very complex, and does not scale well with the number of steps of quantum algorithms or the number of qubits they required. For this reason, smarter mapping techniques are necessary that are able to map these algorithms in a scalable manner and with acceptable overhead, in order to prevent errors. Furthermore, these algorithms require metrics which ought to be optimized: on e of the problems in designing a suitable metric is that quantum devices display very complex noise behaviours, that are typically difficult to capture into a mathematical model and, furthermore, may change wildly across quantum devices and quantum computer technologies.

The main contribution of this work is the development a novel mapping metric that is better able to capture the faulty behavior of NISQ quantum computers. Furthermore, we design an unbiased method (independent of the mapping algorithm) to evaluate the predictive power of our metric and compare with existing mapping metrics.

1.2 Thesis Structure

Thus far, we've highlighted some historical facts and milestones, as well as the main contribution of this work to the field of quantum computation.

In Chapter 2, all the necessary background to the topic at hand is presented. This includes an introduction to Quantum Information Theory, and some of the Physical Implementations capable of storing and processing this kind of information. Then, we delve into an overview of the complete system stack necessary to power a functioning quantum computer. Finally, the mapping problem of quantum computing is highlighted, along with the state of the art solutions aiming to tackle it, and the proposed goals to be achieved in this thesis.

In Chapter 3, new models are proposed, in order to tackle the mapping metrics introduced in Chapter 2. This includes both analytic models analog to the existing ones, but also models using novel techniques such as Deep Learning.

In Chapter 4, having already presented all the theory behind the work at hand, as well as the previous work and novel propositions, the necessary software tools in order to complete the proposed goals are described. This includes both existing tools, and those developed in the context of this thesis.

In Chapter 5, a novel method to evaluate the predictive power of the different metrics is presented, and the different metrics (both existing and new) are evaluated and compared.

Lastly, in Chapter 6, a critical review of the work done is made, and possible future research topics are established.

2

Quantum Computing and the Qubit Mapping Problem

Contents

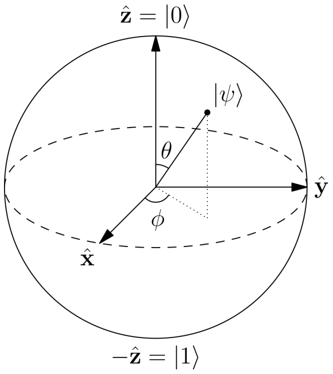


Figure 2.1: Bloch sphere representation of some state ψ. Taken from [4].

In this section, a brief introduction to Quantum Computing (QC) is given, and the Mapping Problem will be stated. In general, QC research covers an entire system stack, from pure quantum algorithms, system architecture, and quantum-chip's low-level physics.

2.1 Quantum Information Theory

2.1.1 Qubits

In Classical Information, the basic unit of information is the bit, which has two possible states, '0' and '1'. It is a deterministic and discrete (binary) variable. In Quantum Information, on the other hand, the basic unit is a quantum bit, or *qubit*, which has two base states, commonly represented as $|0\rangle$ and $|1\rangle$ (known as *computational basis*). However the qubit can also be in a state which is any linear combination of the two basis states, a property called *superposition*. An arbitrary qubit state $|\psi\rangle$ is therefore represented (in a specific basis) as

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.1}$$

where α and β are complex parameters that obey the normalization condition $|\alpha|^2 + |\beta|^2 = 1$.

The space of all qubit states is, therefore, a Hilbert space. In case of a one-qubit state, it is possible to use a geometrical representation called Bloch sphere, as shown in Fig. 2.1, in which a qubit is an unitary vector pointing to any of the directions in the Block Sphere.

Multi-qubit states can be constructed, and are represented by a 2^n dimensional vector, where n is the number of involved qubits. For example, a general multi-qubit state has the following form:

$$|\psi\rangle = \alpha_0\,|000...0\rangle + \alpha_1\,|000...1\rangle + ... + \alpha_{2^n}\,|111...1\rangle \tag{2.2}$$

If the multi-qubit state is non-factorable in individual qubit states, it is said that the state is *entangled*.

2.1.2 Operations

Quantum states are acted upon using Quantum Operations, which are split into three types:

A – State preparations. State preparations change the state of the qubit into a known, useful one. Usually, state preparations leave qubit in one of the computational basis states.

B – Quantum Gates. Quantum gates are reversible, non-destructive, unitary operations applied to qubits. They change the qubit quantum state without making it collapse to the computational basis (hence non-destructible/reversible). They are represented by a $2^n \times 2^n$ matrix operator (n being the number of qubits they act upon) and constitute the bulk of a Quantum circuit model-based computation (explained in the next section). Single-qubit gates act on just one qubit and can be understood as rotations in the Bloch Sphere around a given axis. One example of these is the X gate, which represents a 180º rotation around the $\hat{x}$ axis. Multi-qubit gates, on the other hand, may act on several qubits, and can be viewed as rotations in higher dimensional space. A particularly relevant subset of these are constituted by Two-qubit gates, gates which act on two qubits. An example of these is the CNOT gate, which performs an X rotation in a target qubit if the control qubit is in the $|1\rangle$ state. Fig. 2.1 shows circuit and matrix representations of common quantum gates.

C – Measurements. These operations are needed in order to gather information about a quantum state. However, unlike quantum gates, this operation is destructive, since the quantum state, and therefore the encoded information, is lost upon execution. Measurement makes the quantum state collapses into one of the basis states.

2.1.3 Quantum Circuit Model and Quantum Circuits

The most common Quantum Computing Model is called Quantum Circuit or Quantum Gate model. This type of computations usually ends with measurements, which collapse the resulting quantum state and allow some of the information to be retrieved. It models quantum computation as quantum circuit, which is composed by series of quantum gates on a qubit register. An example of the visual representation of quantum circuits is given in Fig. 2.2.

Quantum algorithms are commonly described as quantum circuits and are hardware-agnostic, i.e. they do not take into consideration possible limitations present in the physical implementation of the qubits (quantum chip). Examples of well-known quantum algorithms are: Quantum Fourier Transform[1], with a $O(N^2)$ time complexity [5], as opposed to $O(N2^N)$ for the classical homologues; Grover algorithm,

[1] This algorithm serves as basis for other famous ones, such as the famous Shor's algorithm.

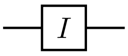
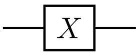
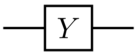
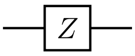
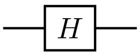
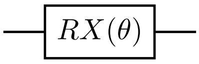
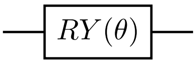
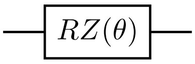
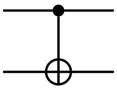
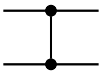
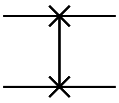
Gate	Representation	Matrix
I		$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
X		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Y		$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Z		$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
H		$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
RX(θ)		$\begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$
RY(θ)		$\begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$
RZ(θ)		$\begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$
CNOT		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
CZ		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$
SWAP		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Table 2.1: Circuit and matrix representation of some of the most common quantum gates. The θ angles are represented in radians. Multi-qubit gates may contain control qubits, represented by the dot symbol •, as present in the CNOT gates. Identity gates (I) also come often in the form of implicit idling gates.

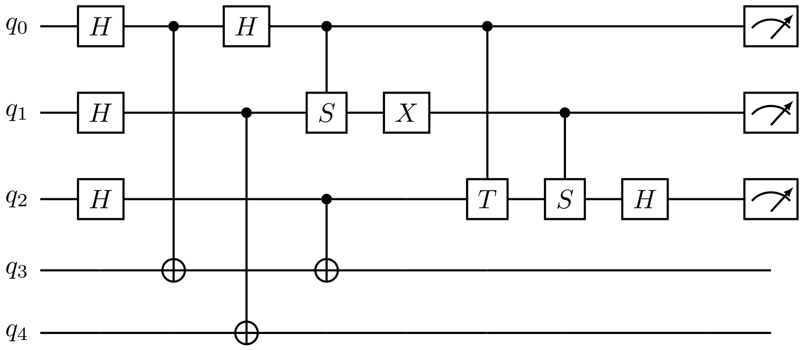


Figure 2.2: Example of a quantum circuit. It contains 5 qubits (q_0 to q_4), single-qubit gates (X, H) and two-qubit gates (S, T, and CNOT, the latter being identified by the $\oplus$ symbol). The meter symbols on the far right represent measurements.

a unstructured search algorithm which provides $O(\sqrt{N})$ complexity as compared to the best possible classical $O(N)$ complexity.

2.2 Physical Implementations

Unlike classical computers, whose implementation mainly relies on silicon transistors, quantum computers have several classes of candidate technologies under research, such as: Superconducting qubits [6], Semiconducting qubits [7, 8], Ultracold atoms/ions [9], Optical [10, 11], Nitrogen-vacancy centers [12], Nuclear Magnetic Ressonance-based [13], and Topological qubits [14]. Within each class there are also several competing technologies, which translate into a myriad of possible qubit implementations. A breakdown of these can be seen in Appendix A.

However, there are three main classes whose maturity stands out, having demonstrated features such as two-qubit gates with high reliability and feasible scalability plans, namely: Superconducting, Semiconducting and Ultracold atoms.

While Superconducting approaches have secured the largest universal implementations, up to 72 qubits [15], other implementations have been accomplishing decisive strides in order to reduce this advantage.

However, due to the elusive and fragile quantum nature of the tasks they aim to perform, quantum devices are especially prone to error. Furthermore due to the different engineering requirements, different implementation technologies and architectures have their own requirements and constraints.

2.2.1 Physical Qubits are fragile, Quantum Operations are imperfect

Quantum systems, such as qubits, are fragile. Noise from the environment can induce unwanted state transitions and phase shifting. To be able to perform coherent computation on the qubits, they must be shielded against this environmental noise.

In most implementations this means placing the quantum chip inside bulky ’fridges’, which cool it to only a few milikelvin. The casing also helps protecting the chip against radiation.

Even without this referred noise, qubits are constructed out of two energy levels, usually from a many-level quantum system. This means that, if the qubit is not in a ground state, it will naturally decay to lower energy levels, a process known as relaxation, limiting the useful lifetime of a qubit state in a fundamental way.

An analogous case happens in classic hardware, with Dynamic Random-Access Memory (DRAM). In DRAM technologies, each bit is stored as a charge level on a small chip capacitor. Over time, the capacitor leaks charge, which would eventually nullify the stored state. To prevent this, extra circuitry periodically reads the bit cell and rewrites it, refreshing the original state. This is not possible in the quantum case, since measurements are destructive. In the quantum case, the average lifetime of qubit states is broadly described as coherence time, which typically ranges from nanoseconds to few minutes [16, 17].

Additionally, the operations applied by quantum devices in their qubits are not the same as the theoretical ones. They are imperfect and may introduce errors in the computation. The best error rates achieved in quantum operations are commonly as low as 10^{-4} for single-qubit gates, and 10^{-3} for two-qubit gates [18]. For the sake of comparison, the error rate of modern classical logic gates is on the order of 10^{-9} [19] and lower, while also being easier to correct.

2.2.2 Qubit Connectivity

In order to perform multi-qubit gates, the relevant qubits need to be physically connected in such a way that it is possible for their quantum wave functions to interfere. This is done in different ways, depending on the implementation. For example, in the case of superconducting qubits this physical connection is often a photonic resonator, whereas in trapped ions the qubit-qubit interaction is mediated via shared motional states imposed by the trap. Qubits connected in such a way are commonly referred to as ’adjacent’. An example of a superconducting chip, along with its respective connectivity grid is present in Fig. 2.3. Fig. 2.7(a) contains examples of connectivity graphs present in other known superconducting chips.

The most commonly implemented type of multi-qubit gate is two-qubit gates. Each of these gates requires a control operand qubit and a target operand qubit. The CNOT gate, for instance, flips the

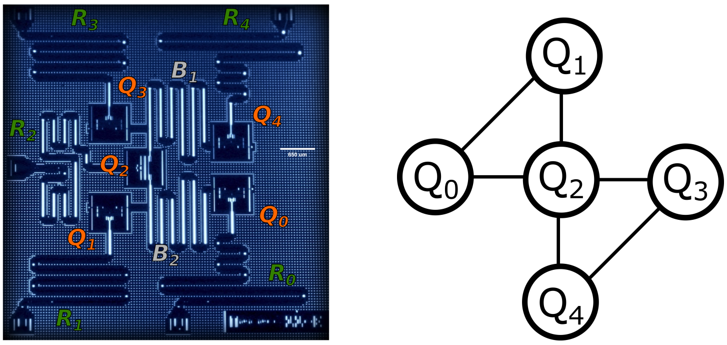


Figure 2.3: Picture of the 5-qubit IBM Tenerife quantum chip [20], along with its respective connectivity graph. Each node represents a qubit, and each represents an interconnection between them.

quantum state (along the x-axis of the Bloch Sphere) target qubit, if the control qubit is the $|1\rangle$ state, and does nothing otherwise.

Unlike the quantum circuit abstraction, quantum chips usually don't have all-to-all connectivity between qubits[2]. That is, the qubit connectivity graph is not a complete graph, since such a feature poses great engineering challenges. Furthermore, complete graph connectivity may not even be completely desirable: this would increase the effects from crosstalk, a phenomenon wherein adjacent qubits incur in unwanted interactions, deteriorating the quality of the quantum states.

The most promising connectivity graph topologies from a feasibility point of view are the 2D lattice and variants thereof. In these types of topology, qubits are connected with the nearest qubits on the chip. This type of connectivity constraint is called Nearest Neighbor.

2.2.3 How to quantify the reliability of the computation?

The error-prone characteristics of quantum computers make the need arise for measures of computation reliability.

One such measure is called *Fidelity*. Given a quantum operation, or a set of quantum operations (such as a quantum circuit), fidelity is a measure of distance in the Hilbert Space, namely the distance between the expected quantum state (assuming error-free computation) and the obtained state. It is given by the square of the inner product of two quantum states, as follows:

$$F(\phi, \psi) = |\, \langle \phi | \psi \rangle \,|^2 \tag{2.3}$$

where ϕ and ψ are two arbitrary quantum states.

[2] A few exceptions apply, such as Trapped Ions in certain conditions. The trapped ions' shared motional states yields all-to-all connectivity between qubits in the trap. Most designs for future scalable trapped ion quantum computers are constituted by several traps containing a small number of ions (typically ≤ 20 ions). In these designs, only qubits within the same trap present all-to-all connectivity. Inter-trap connections are typically assured by ions placed on the edges of each trap, each communicating with the closest ion in the adjacent trap via photons [9].

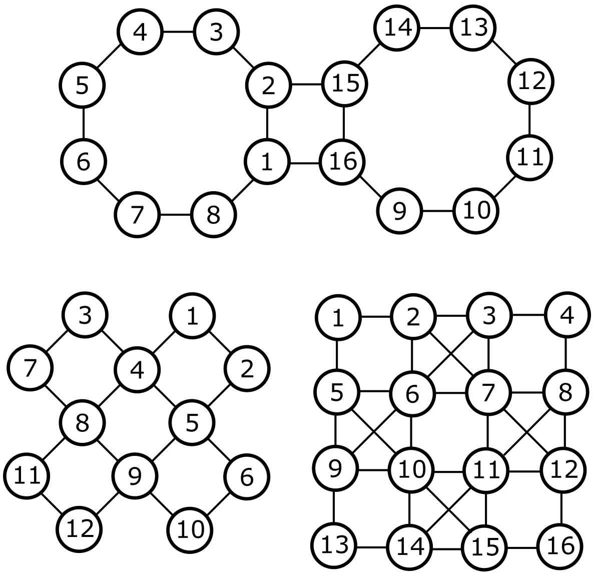


Figure 2.4: Connectivity grid unit cells for the scalable prototypes of different developers: Rigetti (top), Google Bristlecone/DiCarlo Lab (bottom left) and IBM Tokyo (bottom right).

Due to the black-box characteristics of quantum computation, it is not possible to easily inspect and gather all the necessary parameters to describe a quantum state: upon measurement, the quantum state collapses into the computational basis. Instead this inspection is performed via techniques such as tomography, where a quantum state is repeatedly prepared and measured in different bases, in order to reconstruct the state.

Gate Set Tomography (GST) [21, 22] is a technique derived from this principle, which aims to characterise the set of gates a given quantum device is able to perform. The result is a characterization of the (faulty) gates a device is able to execute, from which individual gate *fidelities* can be extracted. This gained knowledge can give useful insights regarding how much a certain type of operation affects the reliability of the computation.

Another popular and intuitive measure of reliability is the *Probability of Success*, or *Success Probability*. This measure represents the probability that, given a computation, the expected result is observed. It is closely related to fidelity, however, instead of measuring the distance between two quantum states, it is typically used to measure the distance between the distributions they each of the states produce, after measurements operations. There isn't a single measure for this probability, in fact, different measures are used in different instances. One deciding factor relies on the distinction between deterministic and probabilistic computations. A *deterministic* computation is one such that, in the absence of errors (perfect operation fidelity), is expected to always yield the same outcome during measurements. This means that, in deterministic computations, no state superposition is present right before measurement. However, a fair share of quantum algorithms are indeed probabilistic: in the same hypothetical error-free conditions, they are expected to yield not a single outcome, but instead a distribution of outcomes.

There are different metrics used for both type of computations:

- **Deterministic:** The probability of success is the fraction of times that the expected outcome (in the absence of errors) is measured, given a sampled distribution.
- **Probabilistic:** The probability of success is given by a given distribution distance measure. Two of the measures used in this context are the Trace/Kolmogorov distance (Eq. 2.4), and the Kullback–Leibler divergence (Eq. 2.5).

$$D(P,Q) = \frac{1}{2}\sum_{x\in X}|P(x) - Q(x)| \tag{2.4}$$

$$D_{KL}(P,Q) = \sum_{x\in X} P(x)\log\left(\frac{P(x)}{Q(x)}\right) \tag{2.5}$$

where P, Q are two discrete distributions, and X is the sample space containing all the possible outcomes.

2.2.4 Quantum Error Correction (QEC) and Fault Tolerance (FT)

Due to the aforementioned qubit frailty and quantum operation imperfections, some error correcting techniques are necessary in order to achieve full-scale and successful quantum computation. However, unlike the case in classical information, arbitrary quantum states cannot be copied, as stated by the No-Cloning theorem [23], and measurements destroy the information held by the qubit. This means that different correction techniques are necessary in the quantum realm. These techniques belong to a domain called Quantum Error Correction [2].

The idea behind the majority of these techniques is to encode quantum states in a redundant way. This is done by encoding single logical qubits using multiple physical qubits, via entanglement. Errors can then be detected by measuring extra ’flag’ qubits, called *ancillas*.

Due to low number of qubits of NISQ devices, such error correction capabilities are, for now, out of reach. Instead, Error Mitigation techniques are being developed [24, 25], which try to tweak the performed quantum operations in order to reduce the amount of introduced noise. These are feasible for NISQ devices, since such methods do not add overhead.

This limited capability to deal with errors in NISQ era gives rise to the need to execute quantum computations in the most efficient manner possible, given a specific quantum device. Such concerns are addressed in the compiler, one of the parts of a working quantum computer system.

2.3 Quantum Computer System Stack

A quantum computer isn't entirely quantum. In fact, it consists of different software and hardware layers that bridge the gap between quantum applications and quantum chips. A Quantum computer system stack proposed by TU Delft is represented in Fig. 2.5 [26, 27].

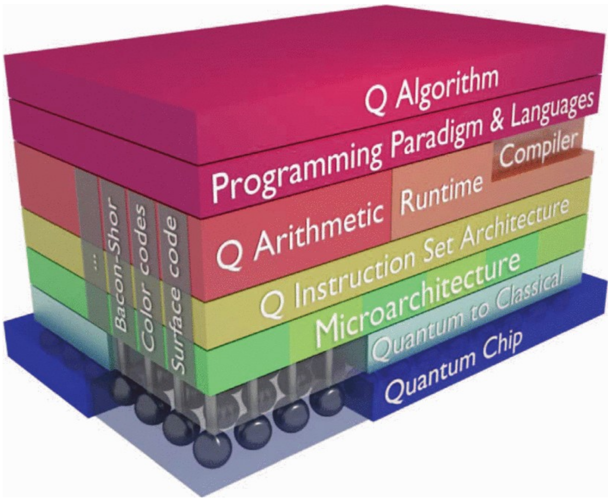


Figure 2.5: A quantum computer system stack, as proposed by QuTech [26, 27].

In decreasing level of abstraction, these are some of the stack parts:

A – Quantum algorithm. Represented using a mathematical description. These typically consist of sequences of abstract unitary operations applied on an unspecified set of qubits.

B – Quantum program. An implementation of a quantum algorithm using a suitable framework. This implementation consists of a set of quantum circuits, which may be interposed by classical control flow. Just like the quantum algorithms that give rise to it, quantum programs too are generally hardware agnostic.

C – Compiler. This is responsible for the ahead-of-time transformation of the quantum program (which was, so far, hardware agnostic) into operations that can be directly performed on the quantum chip. Its functions are described in greater detail in 2.3.1.

D – Microarchitecture. The microarchitecture bridges the hardware-aware code and the hardware itself. It translates that code into a set of operations performed by the Quantum to Classical interface, and includes parts of other functions which have to be completed at runtime, such as QEC.

E – Quantum to Classical. Set of electronics, commanded by the microarchitecture, which allows qubit preparation, control and readout (measurement). It allows control of the quantum state of the qubits via physical electromagnetic, electric or magnetic pulses, depending on the qubits' implementation.

F – Quantum Chip. Lowermost stack level, containing the qubits and their connections to the Quantum to Classical Interface, and their interconnections.

The main objects of study in this work will be two functions of the Compiler part: Initial Mapping and Routing.

2.3.1 Compiler

The compiler is responsible for converting the quantum programs into low-level instructions runnable by the Microarchitecure. These instructions belong to the Quantum Instruction Set Architecture (QISA), which contains the operations that can be directly performed on the quantum chip. These transformations also intend to make the circuit to make it efficiently executable on the quantum processor. It performs the following functions.

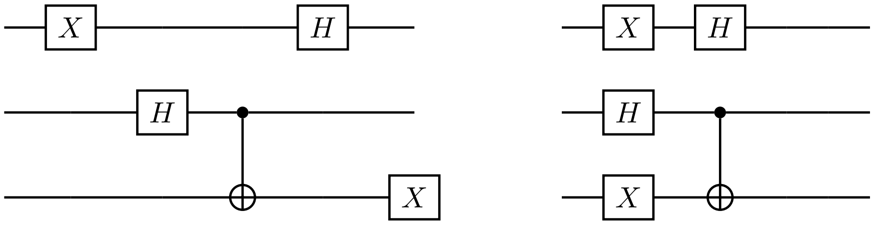
2.3.1.A Gate Decomposition

Quantum chips only support a restricted set of gates. Since quantum circuits can contain any arbitrary quantum gate, it is necessary to decompose these into the hardware's native gate set, also known as *primitives*/*primitive gates*, i.e. gates that the hardware can execute physically. If this gate set is universal, then any arbitrary gate can be decomposed into the elements of the latter.

2.3.1.B Scheduling of Quantum Operations

This process generates a Directed Acyclic Graph (DAG) that encodes the order dependency between quantum operations. A good generator also takes into account that some operations are commutative, and is able to add this information. This can be important since it offers more optimization options.

Furthermore, the scheduler can also schedule a quantum circuit, assigning to each gate a specific timestamp, besides imposing an order relationship. In most physical implementations, where the gate duration differs among the gates but where they are all multiples of a shortest non-zero gate duration, that shortest duration is known as a clock *cycle* [28], or *moment* [29]. Gates executed in the same cycle are known as *parallel*, and typically appear in the same horizontal position (vertically adjacent) in time-scheduled quantum circuits. A good scheduler should be able to able to maximize the parallelism of quantum gates (as exemplified in Fig. 2.6), which in turn results in shorter computation times and lower error rates.



(a) Unscheduled circuit.

(b) Scheduled circuit.

Figure 2.6: Depiction of the greater gate parallelism enabled by a scheduling procedure when applied to an example circuit.

2.3.1.C Mapping

The Mapping procedure deals with the quantum chips' limited connectivity problem, described in Sec. 2.2.2, by smartly assigning (mapping) the virtual qubits to the physical qubits, and applying transformations to the input quantum circuit (which dynamically changes the mapping along the computation) such it becomes compliant with the chips' qubit connectivity graph.

This procedure is the topic in focus for this work, and is described in Sec. 2.4.

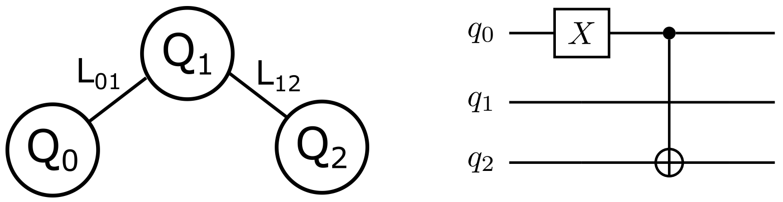
2.4 The Mapping Problem

In most implementations, the mapping problem tends to be divided into two subproblems: Initial Placement (also known as Qubit Allocation) and Routing (also known as Qubit Movement).

2.4.1 Initial Placement

The Initial Placement procedure maps the quantum circuit's virtual qubits to the quantum chip's physical qubits, before the execution of the computation. As such, it creates an initial mapping of virtual-to-physical qubits, which may then be dynamically changed during the computation via Routing (Sec. 2.4.2). A smart initial-placer places interacting pairs of virtual qubits in adjacent (connected) physical qubits.

An example of this is presented in Fig. 2.7: take into account the sample device's qubit connectivity graph present in Fig 2.7(a), where we intend to run the quantum circuit present in Fig. 2.7(b) (which we will assume to be already decomposed into primitives and scheduled). Before running, one must assign



(a) Example of a connectivity graph. Nodes represent the physical qubits and edges their connections.

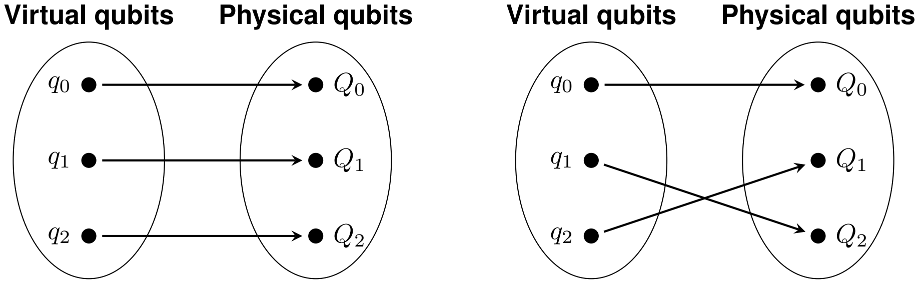
(b) Example of quantum circuit that presents a single-qubit gate on q_0 and a two-qubit gate on q_0 and q_2.

Figure 2.7: Example of a quantum chip physical qubit graph and a quantum circuit which it cannot run without modifications.

the virtual qubits (q_0, q_1, q_2) to the physical qubits (Q_0, Q_1, Q_2) in the quantum chip. The most trivial mapping would be the one present in Fig. 2.8(a), a pure index-to-index correspondence.

It can be seen that the circuit is not executable on the device. While the X gate, being a single-qubit gate, poses no problems, the CNOT is not executable: the CNOT was to be performed (according the mentioned mapping) between physical qubits Q_0 and Q_2, which do not share a physical connection, rendering the gate unfeasible.

Nevertheless, a smart initial mapping, such as the one present in Fig. 2.8(b), allows the execution of the presented circuit, since physical qubits Q_0 and Q_1 are adjacent.



(a) Trivial mapping.

(b) Custom mapping.

Figure 2.8: Examples of possible virtual-to-physical qubit mappings for the device present in Fig. 2.7(a).

2.4.2 Qubit Routing

While initial placement is a relevant procedure, it cannot make all circuits runnable by itself. Even with a good initial mapping, some (or most) of the multi-qubit gates of quantum circuit might not be directly executable, if the connectivity constraints are not respected during the gates' time of execution.

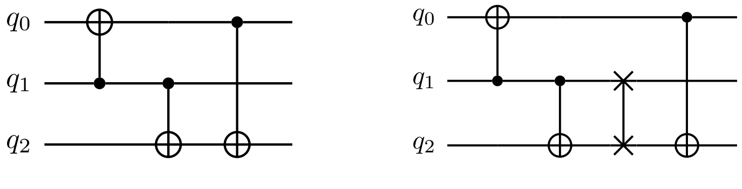
The solution is to dynamically change the mapping during the computation. This procedure is called *Routing*, since usually it is conceptually done by moving an already initialized qubit state through the qubit graph. The qubits can be moved via different methods:

A – SWAP-based methods The SWAP gate is a two-qubit gate that exchanges the states of their operands. Applying a finite number of SWAPs it is possible to route any two virtual qubits through the physical qubit graph such that, in the end, they are in adjacent physical positions. Once this is done, the connectivity constraint is respected and a two-qubit gate may be performed between the virtual qubits.

Other SWAP-based schemes are possible [30], such as SWAP with ancillas, SWAP with lower logical depth, Rotation and Reversal.

B – Teleportation-based methods Teleportation can be conceptually understood as a long range swap. It relies on the creation EPR-pairs, pairs of qubits that share a maximally entangled state. However, this procedure usually requires the usage of measurement operations, which are especially slow (sometimes more than 10 times slower than gates), and faulty. The creation of a high-fidelity EPR-pair is very computationally expensive and, additionally, they have to be distributed to the desired places before usage. This makes the usage of teleportation mainly unfeasible in NISQ devices which, as aforementioned, have especially short coherence times and faulty operations. These type of qubit movement procedure is therefore not considered within this thesis.

An example of routing via the SWAP-based method is presented: suppose that the circuit displayed in Fig. 2.9(a) is to be executed in a device with the connectivity graph present in Fig. 2.7(a). In this case, there is no single mapping that makes all gates constraint-compliant. The mapping present in 2.8(a) satisfies the constraints for the first two gates, while the third gate is not executable. This can be solved by introducing a SWAP gate in the third position, as displayed in Fig. 2.9(b). The effect of this will be to dynamically change the virtual-to-physical qubit mapping, transforming it into the one present it 2.8(b) from the third cycle onwards. The operands of the last gate (q_0 and q_2) are now assigned to adjacent physical qubits (Q_0, Q_2), meaning that the gate becomes executable.



(a) Unrouted circuit.

(b) Routed circuit.

Figure 2.9: Example of a routing scenario in order to make the circuit executable in the architecture present in 2.7(a), via the usage of a SWAP gate.

2.4.3 The importance of mapping overhead optimization

As aforementioned, quantum computers are especially prone to errors, due to the limited lifetime of qubits states and faulty quantum operations. The routing procedure also contributes to this error, by introducing overhead in the form of additional SWAP gates and increasing the execution time of the circuit, since those extra gates in the within the circuit might delay posterior gates. A lower quality initial placement may also introduce overhead, by increasing the need for routing.

Given that NISQ devices are especially faulty, and are less able to implement error correction methods, it is of interest to reduce this overhead as much as possible, in order to maximize the probability of success of the computation. That is, among the different routing possibilities, the goal is to choose the one with the least overhead. Since our main interest is to maximize the probability of success of the computation, one would think that this should be the metric to maximize.

As it turns out, however, the probability of success is not an easily modelable and predictable metric: first because the probability of success intrinsically depends on the noise behaviour of the specific device in question; and second because noise models are very difficult to construct, even for a specific device, due to the complexity and black-box behaviour of quantum systems.

For this reason, the metrics of choice to be optimized during the routing process have been simpler quantities which are known to be correlated with the probability of success, namely:

- **Circuit length/depth**: Number of timesteps of a circuit;
- **Circuit size**: Total number of gates;
- **Variants of circuit size**: such as number of two qubit gates.

All of the above are negatively correlated with the probability of success, that is, the higher the circuit depth/size, the lower the probability of success of the algorithm.

Once a metric is defined, it is possible to build mapping algorithms that try to minimize it, or use general purpose minimization algorithms in order to find the best possible routing scheme, guided by the metric.

The goal of this thesis is to develop better metrics and models which allow the prediction of the relative probability of success of quantum circuits.

One of the main issues when solving the mapping problem, is that it is hard from complexity-theoretical point of view. How hard? From a Graph Theory standpoint, Initial Placement can be viewed as an instance of the Subgraph Isomorphism problem [31], which is known to be NP-complete [32]. On the other hand, Routing can be seen as an instance of the Token Swapping problem [31], which is NP-complete [33] as well.

This means that attempting to find an optimal solution for either problem tends to quickly become an insurmountable problem for problems containing a growing number of qubits and gates.

2.5 State of the Art

2.5.1 Routing Algorithms

Due to complexity aspects, finding an optimal solution to the qubit mapping problem quickly becomes an impossible task with a growing number of qubits and gates. For this reason, despite exact solvers like SMT-based [34] [35] having been proposed for very small instances ($\lesssim 5$ qubits), scalable algorithms are required. In order to tackle this scalability problem and make bigger instances tractable, two methods have been employed:

- **Compartmentalization** Split the problem into smaller, more digestible problems. This is typically done at the quantum circuit level, by slicing it into smaller sections, containing a subset of the gates, which are more easily optimizable (using exact techniques or otherwise). While this method allows for scalability, it might reduce the quality of the resulting mapping, since instead of looking for a minimum of the whole circuit mapping problem, we are looking for minima (global or otherwise) for each of the subproblems.

  This method may also feature look-back and look-ahead windows. These techniques essentially enlarge the scope of the section: they introduce some knowledge of the past in order to allow better interleaving of new routing operations with the previous ones; and they prevent wasteful routing operations by considering future gates and their constraints.

- **Usage of approximate/heuristic solvers** Another option (non-mutually exclusive with the above) is to use approximate/heuristic solvers, which do not aim to find a global optimum, but instead an approximated one.

The majority of proposed solutions implement handcrafted algorithms specialized not only for the mapping problem, but all the way down to the specific quantum technology and micro-architecture. Given the higher maturity and readiness of superconducting devices, most of the mapping literature focus on these, especially on IBM quantum devices [36–38], due to their cloud availability.

One of these higher profile proposals is the algorithm by Zulehner et al [39], which looks for optimal mappings using the A* search algorithm, and then computes the necessary qubit movement between mappings, in what is known as the Mapper-Permutter framework (see Annex B for a more detailed discussion on this). Another example based on the same framework is the one by Lin et al [40], which uses Spectral Graph Theory in order to discover the mappings (although currently limited to a 1D architecture). Other proposals ditch the optimal mapping step; these frame the problem in terms of finding the optimal SWAP placements such that every gate becomes connectivity compliant (see Annex B for a deeper discussion on this) [28, 30, 31, 41]. These handcrafted algorithms employ techniques such as fixed/variable look-back/look-ahead techniques, which tend use search windows of up to 50 cycles.

Some practice reversed mapping [41], making use of the reversibility property of quantum gates in order to attempt higher quality mappings.

Other attempts use an Operations Research approach, applying techniques such as Constraint Programming and Temporal Planning [42–44] (based on the SWAP framework). These usually rely on describing the mapping problem (on a per-device basis) as a mathematically model capable of being read by one of the many off-the-shelf, time-tested solvers, ditching the need for handcrafted algorithms for each technology/microarchitecture. While designing the required mathematical model may arguably be simpler than designing a specialized algorithm, it may also be argued that specialized algorithms have the potential to be faster and better. However, the current literature lacks benchmarks coherently comparing the two approaches.

Furthermore, approaches involving the use of machine learning techniques such as reinforcement learning have also been published [45], albeit with scalability concerns and inconclusive results.

Finally, some works extend the mapping problem, by suggesting algorithms that work at the logical qubit level, enabling scenarios where quantum error correction is used [46].

2.5.2 Metrics used in the Mapping Problem

As referred in Sec. 2.4.3, the main metrics used to guide the mapping procedure are circuit depth and circuit size, due to the daunting task of modelling the probability of success.

However, circuit depth and circuit size may present poor correlations depending on:

1. Device characteristics, such as the fidelity of each gate type;

2. The type of circuit, since different circuits may have different structures that affect the probability of success in ways that the circuit size and circuit depth alone cannot capture, due to the way errors are propagated among the qubits.

Developing a metric that takes device calibration data (e.g. gate fidelities) and the circuit's structure can be a worthwhile effort, since it may be a better predictor of the Probability of Success, whose maximization is the main goal due to the shortcomings of NISQ devices. A better metric may be more likely to correctly guide optimization algorithms to such an outcome.

Some recent forays have attempted to solve this issue by introducing some operation fidelity-aware methods [36, 47–49]. An approach which both formalizes and generalizes the so far proposed methods in the literature is the Estimated Success Probability (ESP), as defined by Nishio et al. [36]:

$$ESP(C) = \prod_{g \in C} (1 - \epsilon_g) = \prod_{g \in C} (f_g) \tag{2.6}$$

where C is an arbitrary quantum circuit, g a gate, ϵ_g the error rate associated with gate g, and f_g the corresponding fidelity. The fidelities can be extracted from the device's calibration data.

This approach goes one step further and does not consider a simple average fidelity for each type of gate, but rather takes into consideration the issue of hardware variability: different qubits and different qubit connections have divergent fidelities associated with the quantum operations they enable, due to imperfect fabrication procedures. The knowledge about this variability can be used to reduce the errors introduced by the mapping overhead. This can be done by, for example, choosing routing schemes that favour the usage of higher fidelity links in order to construct the qubit routes.

However, the existing proposals tend to suffer from a couple of issues: firstly, the metrics only take into account the performed two-qubit gates, and don't consider the impact of single qubit gates and the implicit idling gates; secondly, is that the proposed metrics are typically defined and benchmarked in the context of a specific mapping algorithm. The latter, as will be argued, is not ideal, as it ambiguates the merit of the metric with that of the algorithms that employ it, i.e. it may be hard to attribute any good or bad performances of mapping procedure to the algorithm, to the metric, or to the synergy of both (i.e. due to the day the algorithm explores the given metric).

Due to the lack of research done mapping metrics, and the mentioned issues, the goals for this thesis are:

- To test the performance of the existing metrics, namely circuit size, circuit depth, number of two-qubit gates and Estimated Success Probability (ESP) [36];
- Develop higher quality metrics.
- Develop a new method to compare the mentioned metrics that is able to rate the merit of each one in a algorithm-independent manner.

3

Metrics Models

Contents

As referred in the State of the Art section, the only metrics so far used in literature to attempt to predict the relative success probabilities of circuits were the circuit's depth, size, number of two-qubit gates and ESP (which actually attempts to model the probability of success).

In this section, new models of the success probability are developed, which are tested alongside the previously attempted ones in Chap. 5. Namely, four variations of the ESP model are proposed, along with other analytical models that also make use of the reliability data of devices such as gate fidelities. Additionally, we propose new deep learning-based methods that learn to predict the probability of success of quantum circuits when presented with a labeled circuit database. For all models, it should be noted that:

- Qubit initializations are assumed to be perfect;
- Measurement operations are not included since running the same circuit thousands of times in order to obtain the output distribution would be computationally expensive, due to the fact that a simulator was used (as discussed in Chap. 4).
- CZ was the considered two-qubit gate when developing the models, since the native two-qubit gate that the backend supports (Chap. 4).

3.1 Analytical metrics/models

3.1.1 Estimated Success Probability (ESP)

ESP was the first model that was non-device agnostic, in the sense that it aimed to take the device's specific error rates into account. In order to evaluate the ESP models at scoring different circuits, we define four variants inspired by Eq. 2.6. The difference between these variants is the way individual gate fidelities are determined and how implicit elements of the quantum circuits are accounted for.

- **ESP^0:** This is the variant implemented in Nishio et al [36]. It considers errors provenient solely from **explicit** single- and two-qubit gates. As such, it does not take into account the implicit idling gates – that is, the waiting periods during which the qubits are not active as a result of scheduling. Additionally, instead of defining fidelities on a per-gate basis, a single fidelity values is defined for every gate within each gate type (gate types referring to whether each gate is a single- or two-qubit gate). The result is that single-qubit gates are all assigned the same fidelity, irrespective of the specific rotation they perform. This does not influence the way two-qubit gates are processed, since typical quantum computer realizations only implement a lone two-qubit gate. Considering the fidelities on a per-gate type basis is a reasonable simplification of the circuit's information, since the fidelity variation for gates within same type is usually much smaller than the fidelity variation between different gate types.

- **ESP^{i}:** Similar to ESP^{0}, but also takes into account the impact of the **implicit idling** gates.

- **ESP$^{0}_{sr}$:** Similarly to ESP^{0}, this variant does not consider implicit idling gates. However, it accounts for the information regarding the **specific rotation** of each gate, instead of solely considering the gate type. Although making no difference in the two-qubit gates case (given that typically a lone two-qubit gate is implemented per device), this approach spares individuality of each of the single-qubit gates, i.e. the information regarding the specific rotation is maintained, instead of agglomerating all single qubit gates under the same umbrella. The result is that single-qubit gates are not assigned a common (averaged) fidelity value. Instead, each single qubit gate is assigned the fidelity for the specific rotation they perform.

- **ESP$^{i}_{sr}$:** Considers both the **implicit idling** gates and the **specific rotation** of each gate.

While the ESP^{0} variant may arguably less interesting due to the fact it does not consider idling, it was still considered for this work in order to enable comparison with the results obtained in Nishio et al [36]. In order to slowly build up on top of this variant, ESP$^{0}_{sr}$ was also defined and evaluated. However, for the models proposed in the following sections, implicit idling gates are always considered.

Along with the proposed ESP variants, three novel analytical are also proposed, as described in the following section.

3.1.2 Tracked Estimated Success Probability (TESP)

The ESP models take into account device specific data such as gate fidelities, which was not the case for the previous metrics (circuit depth, circuit size and number of two qubit gates). Two additional models, named TESP1 and TESP2, where TESP stands for Tracked Estimated Success Probability, were developed using a different perspective and assumptions assumptions regarding how errors affect and propagate through quantum circuits.

These models attempt to track reliability on a per-qubit basis, until the end of the circuit where these can be merged into a single success probability value. In these models, it is assumed that the qubits are flawlessly prepared in the computational basis (usually in the $|0\rangle$ state), i.e., their initial fidelity is 1. The decoupled, per-qubit reliability after idling or a single-qubit gate is calculated for both models through the following expressions:

$$\text{TESP}_q^0 = 1.0, \quad \text{(Initial prob. succ.)} \tag{3.1}$$

$$\text{TESP}_q^{t+1}, = \text{TESP}_q^t, \times f_{idle} \quad \text{(Idling gates)} \tag{3.2}$$

$$\text{TESP}_q^{t+1} = \text{TESP}_q^t \times f_{1qbg} \quad \text{(Single-qubit gates)} \tag{3.3}$$

where TESP$_q^t$ represents the reliability of an arbitrary qubit q at timestep t and f_{idle}, f_{1qbg} represent the reliability of idle- and single-qubit gates, respectively. As for the two-qubit gates, the rules differ for each of the models:

$$\text{TESP1}_{q_c}^{t+1} = \text{TESP1}_{q_c}^t \times f_{2qbg}, \tag{3.4}$$

$$\text{TESP1}_{q_t}^{t+1} = \text{TESP1}_{q_t}^t \times f_{2qbg}, \tag{3.5}$$

$$\text{TESP2}_{q_c}^{t+1} = \text{TESP2}_{q_t}^{t+1} = \text{TESP2}_{q_c}^t \times \text{TESP2}_{q_t}^t \times f_{2qbg} \tag{3.6}$$

where q_c (q_t) represents the control (target) qubit, and f_{2qbg} represents the reliability of a two-qubit gate. For these models, reliabilities are considered solely on a per-gate type basis. The final step is merging of the per-qubit reliabilities into a final success probability to be used as the models' score:

$$\text{TESP}^T = \prod_q TESP_q^T \tag{3.7}$$

that is, the *TESP* value for a given circuit is the product of the values for each qubit at the last timestep (timestep T).

As can be seen from the expressions above, the TESP models differ on the assumption used to model the effect of two qubit gates. TESP1 assumes two-qubit gates to impact each of the operands equally, i.e. they both receive an error that is relative to the gate's fidelity and their reliability value. TESP2, on the other hand, attempts to assume the worst possible case of the effect of gates on qubits, making the resulting reliability of each operand a result of not only its initial reliability and the gate's fidelity, but also of the other operand's reliability. This model, based on [50], aims to reflect the worst case effect of gates (including idling gates) on the reliability of the qubits.

The TESP models aim to treat the reliability of the computation as a local quantity (that is, dependent on each qubit), by considering the local impact of errors that the circuit's gates yield, using therefore more of the information carried by the quantum circuit which might prove useful in some instances. Some cases where ancillas (auxiliary qubit) are used: if the final reliability of ancillas does not matter, such as cases where qubits are solely used for routing at some point during the computation and are left unused afterwards, then their final reliability can be discarded, since it was already being independently tracked. These use cases are however out of the scope of this work.

In TESP1 case, the expressions can be implemented as they are. In the TESP2 model case, however, the displayed set of expressions for this model should not be directly implemented in a computer program. This is due to a very fast scaling in the order of magnitude of the outputted fidelity: the recurrent multiplications of values lesser than 1 introduced by two-qubit gates may quickly exhaust the floating point precision of the computer. In order to tackle this issue, the expressions are implemented in logarithmic space, using the expression $\log(a \times b) = \log(a) + \log(b)$. These allow us to replace the recurrent multiplication of very small numbers by the sum of more manageable numbers. The set of implemented expressions for TESP2 is:

$$\text{TESP2}_q^0 = 0, \quad \text{(Initial value)} \tag{3.8}$$

$$\text{TESP2}_q^{t+1} = \text{TESP2}_q^t + \log(f_{idle}), \quad \text{(Idling gates)} \tag{3.9}$$

$$\text{TESP2}_q^{t+1} = \text{TESP2}_q^t + \log(f_{1qbg}), \quad \text{(Single-qubit gates)} \tag{3.10}$$

$$\text{TESP2}_{q_c}^{t+1} = \text{TESP2}_{q_t}^{t+1} = \text{TESP2}_{q_c}^t + \text{TESP2}_{q_t}^t + \log(f_{2qbg}), \quad \text{(Two-qubit gates)} \tag{3.11}$$

where it should be noted that the initial value is 0, since $\log(1) = 0$.

While the usage of logarithmic space allowed added some numerical resilience to the model, it was still not capable of handling circuits with more than 20 two-qubit gates. Fixing this required a second trick: the TESP2 value for each qubit was represented not in the default *double* floating point number, as before, but rather in an arbitrary precision type. The specific type used was *cpp_dec_float* from the popular C++ Boost Libraries. This new representation for the floating point values allowed the model to perform for circuits well beyond 2000 two-qubit gates.

3.2 Deep Learning-based models (NNESP)

Besides the presented analytical models, attempts were made in this work to develop better models of the success probability. Namely, data-based models were especially enticing, due to the complexity and variability of quantum devices: noise models can vary wildly even between devices from the same technology class. Data-based models may have the ability to learn these behaviours on a per-device basis, saving development time and costs.

One particularly promising approach is the usage of Deep Learning (DL), which has in the last years taken the AI world by storm due to its ability to learn complex problems, such as Image/Voice

Recognition, discover novel materials and drugs, among others. Deep Learning uses a mathematical construct called Neural Network (NN) for which [51] is a recommended source. Usually deep learning approaches pose a couple of challenges:

A – Training requires large amounts of data. For the problem at hand, it is possible to generate an arbitrary number of random quantum circuits (input data) and run them in the quantum device in question in order to retrieve their probability of success (output data/labels). Since data is readily available, we can use a supervised learning paradigm.

B – Models lack explainability. For the purpose of rating equivalent circuit alternatives, our problem does not require explainability.

Given that our problem is suitable to be tackled by deep learning approaches, some common questions remain:

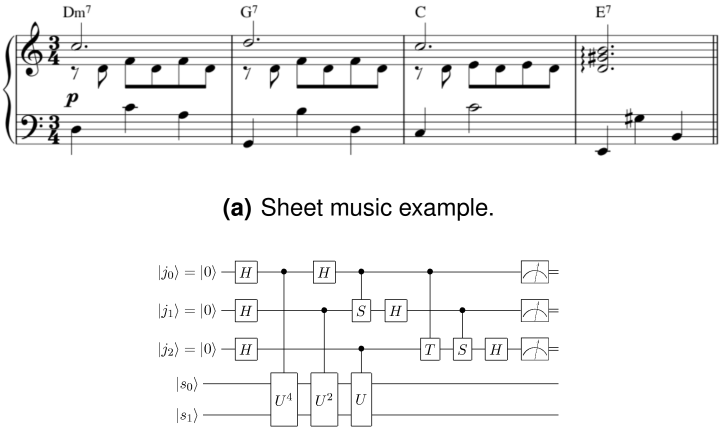
- What is the best representation for the data?
- Which neural network architecture can best capture the behaviour of the data and make the most accurate predictions?
- Which loss function best translates the quantity to be minimized during training?
- How well can the model perform for data it has not seen before?
- How can the training process be sped up?

These questions shall be addressed in the following sections

3.2.1 Type of Data and Neural Network Architecture

Our goal is to predict the probability of success of any quantum circuit we provide. Therefore, in a supervised learning setting, we wish to provide the quantum circuits as input data, and the obtained probabilities of success (from device experiments or simulation) as output data. In order to design a suitable architecture for the problem, we must consider the type of data that we will be processing.

Quantum circuits are essentially time-ordered schedules where each operation is assigned to a specific qubit on a specific timestamp, since the order of the operations matters and cannot be ignored, so quantum circuit structures can be seen as sequential data. In fact, quantum circuits are not visually dissimilar from other types of sequential data, for example, music. In this kind of sequential data, the timestamped operations are musical notes (analog to quantum gates), which are assigned a specific tone (analog to qubits, in the case of quantum circuits). Furthermore, these sequential data types can be trivially divided in timesteps, by using the duration of the shortest operation as the basic unit of time: cycles in the case of quantum circuits, and the shortest note duration in the case of music.


Dm^7
G^7
C
E^7
p
$|j_0\rangle = |0\rangle$
$|j_1\rangle = |0\rangle$
$|j_2\rangle = |0\rangle$
$|s_0\rangle$
$|s_1\rangle$
H
S
T
U^4
U^2
U

(a) Sheet music example.

(b) Quantum Circuit example.

Figure 3.1: Visual comparison between sheet music (snippet from *Fly me to the Moon*, by Frank Sinatra, taken from [52]) and a quantum circuit (taken from [53]).

Such conceptual similarity served as inspiration for the proposed architecture in this work. Music and other kinds of sound data are typically processed in a Deep Learning setting using Sequential Models such as Recurrent Neural Networks (RNNs). Using these, it's possible to, for example, extract emotional connotations [54] or other quantities of interest from music.

3.2.2 Data Representation

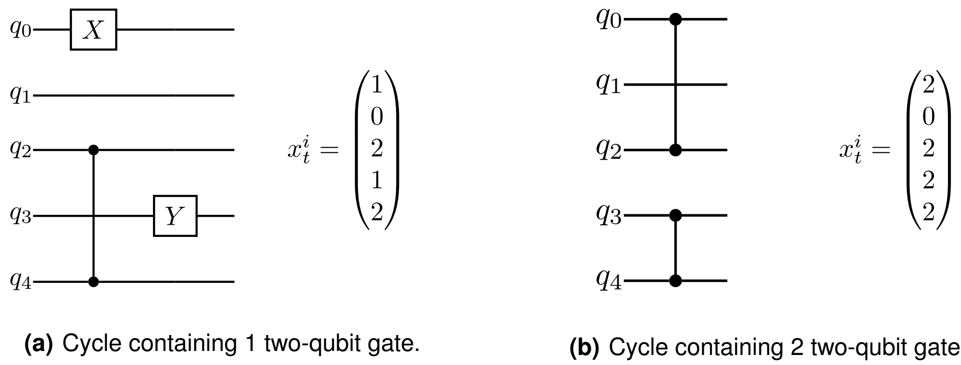
Now type of data is identified, we now have to encode it in a way that a neural network is able to receive and process. For each circuit, the procedure is to encode each circuit cycle as a feature vector. Each feature vector will serve as data timesteps to be fed to the neural network.

One possible decision is to simplify the encoding by dropping some potentially less important information. One such possible simplification is to discard the specific function of each gate, and just keep instead the type of the gate, just like it was done in the analytical models except for the ESP^0_{sr}/ESP^i_{sr} cases. In a deep learning context, such simplification may be acceptable and interesting due to the following reasons:

- This allows the encoding the quantum circuit data in a concise and fully categorical manner, allowing a better scaling with the number of qubits;
  - Only having to specify the gate type, and not rotation parameters, namely axis and angles. This results in a reduced number of required model parameters;

- The gate type is believed to be a strong indicator of the impact of a specific gate, since gate fidelity does not vary a lot within the same time, compared to the fidelity difference between different types. In fact, the same strategy was used for most of the analytical models (except ESP$^{0}_{sr}$/ESP$^{i}_{sr}$);
- Due to the ability by deep learning to model complex behaviours, the obtainable results might be useful to form an educated guess of the prediction accuracy's upper bound that obtainable considering gate types only (i.e. the same fidelity for all gates of the same type). This is interesting, given that the majority of the previously proposed metrics (ESP, depth, size, number of two-qubit gates, etc) depend on information that can be extracted from these gate type-only quantum circuits.

This simplified gate type-based approach, henceforth denominated as NNESP (standing for "Neural Network-Estimated Success Probability"), which considers the implicit idling gates but not the specific rotations, will be used in order to introduce the proposed circuits encoding procedure. Afterwards, the case where, similarly to the ESP$^{0}_{sr}$/ESP$^{i}_{sr}$ models, no gate rotation's information is discarded is also presented. This approach is denominated by NNESP$_{sr}$, where the sr index stands for specific rotation. Due to the simulation speed constraints, 5-qubit quantum circuits for the test of all models. Additionally, the two-qubit gate that is implemented in our model is be the CZ (Controlled Z, also known as Controlled Phase, or CPHASE), since it is the primitive gate of the device that the backend attempts to model. This allows for further simplification, since Controlled Z (CZ) is operand-symmetric, i.e. no directionality (control, target) needs to be specified. For the NNESP case, a naïve representation of a 5-qubit circuit would be, for each timestep, a vector of features where each feature is the gate type each qubit is undergoing (encoded using the rules in Tab. 3.1), as exemplified in Fig. 3.2.


q_0
q_1
q_2
q_3
q_4
X
Y
$x_t^i = \begin{pmatrix}1\\0\\2\\1\\2\end{pmatrix}$
q_0
q_1
q_2
q_3
q_4
$x_t^i = \begin{pmatrix}2\\0\\2\\2\\2\end{pmatrix}$

(a) Cycle containing 1 two-qubit gate.

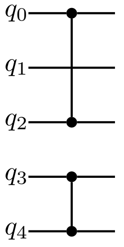
(b) Cycle containing 2 two-qubit gates.

Figure 3.2: Examples of circuit cycles and their corresponding timestep encoding, according to the rules displayed in Tab. 3.1. x_t^i represents the input feature vector at timestep t of the i-th circuit.

We can see, however, that this naïve representation loses information, as is evident in Fig. 3.2(b): In cases where more than one two-qubit gates are present in the timestep, information is lost regarding pairs of operand assigned to each two-qubit gate. A straighforward way to differentiate each pair of

Type of gate	Feature label
Idle	0
Single-qubit	1
Two-qubit	2

Table 3.1: Naïve integer encoding for the NNESP model.



$$x_t^i = \begin{pmatrix} 2 \\ 0 \\ 2 \\ 3 \\ 3 \end{pmatrix}$$

Figure 3.3: Example of an improved encoding for the NNESP model which, unlike the naïve encoding in Tab. 3.1, maintains a clear association between each symmetric two-qubit gate and its operands. For asymmetric two-qubit gates like the CNOT, another encoding would be necessary not to the directionality information.

two-qubit gate operands is to use a bigger feature dictionary, as presented in Tab. 3.3. According to these new rules, the same unique label is assigned to all operands of the same gate and no information regarding the parent two-qubit gate for each operand is lost.

Furthermore, it's usual to forgo the usage of 0 as an encoding label, since its usage is generally reserved for padding, as will be explained shortly. Therefore, a more useful integer encoding is the one displayed in Tab. 3.2.

Type of gate	Feature label
Idle	1
Single-qubit	2
i-th Two-qubit gate (in cycle)	$i+2$

Table 3.2: Final integer encoding for the NNESP model, built upon from the encoding in Tab. 3.3. This encoding foregoes the information of the specific rotation performed for each single-qubit gate, treating them all as common entity. Unlike the encoding in Tab. 3.1, this is a suitable encoding in the case that CZ (CPHASE) or other operand-symmetric gate is the lone native two-qubit gate.

As aforementioned, we also propose an approach that, similarly to the ESP$_{sr}^{i}$ approach, distinguishes between different instances of single-qubit gates. This ”upgrade” only requires relatively simple changes to the scheme in Tab. 3.2: creating additional labels for each of the single-qubit gates, and offsetting the two-qubit gate labels by such an amount (Tab. 3.3). No extra labels are required to differentiate between two qubit gates, since most (if not all) quantum chip implementations only provide a single native two-qubit gate (typically CNOT or CZ for superconducting qubits). When targeting devices whose two-qubit gate is asymmetric (such as CNOT), a different encoding which can differentiate between the control and target operands should be devised. Similarly, when developing an encoding for devices that support arbitrary single-qubit gates, an alternative encoding is also needed.

Type of gate	Feature label
Idle	1
Single-qubit	$[2, 1 + N_{single}]$
i-th Two-qubit gate (in cycle)	$i + 1 + N_{single}$

Table 3.3: Lossless integer encoding used for the NNESP$_{sr}$ model, in the case that CZ (CPHASE) or other operand-symmetric gate is the single native two-qubit gate and the set of single-qubit gates is finite. N_{single} represents the number of discrete single-qubit gates considered.

The encoding used by the NNESP$_{sr}$ can be seen as lossless, since no information regarding the circuit is lost: the gates and respective are not ambiguously defines, and the circuit's structure (the specific distribution of the gates throughout the circuit) is unequivocally defined via the feature vectors' indexes.

A – Padding and Masking Additional steps are required before having an implementation-ready encoding: padding and masking. Before fully addressing this topic, a detour must be made regarding one of the implementation aspects of the training of NNs: batch training.

When training NNs, the number of training examples that are fed per iteration is called the batch size. Each of these examples within the batch can be typically processed in parallel, depending on the hardware's capabilities. Here, processed means that the forward propagation is computed in parallel, a training loss is computed for the entire batch (instead of just a single sample), and then backpropagation updates the weights of the network. Batch training allows multiple computations to be done and parallel, greatly speeding up the training process.

However, in the case of RNNs, an additional rule applies: all the training examples need to have the same number of timesteps. In order to do this, we increase the length (number of timesteps) of the training examples within the batch, so that they match the length of the longest example. This is done by appending null information (vectors filled with zeros) to the shorter examples (Fig. 3.4), hence justifying the decision of not encoding idle gates with a zero-label. While padding incurs in a memory penalty, the performance penalty can be minimized: the NN does not have to take into account the null (0-filled) timesteps in the computation. The performance penalty can be avoided by simply placing a Masking Layer as the first layer of the network (before the embedding layer).

B – One-hot encoding Only one alteration to encoding is now due: to transform the feature vectors from integer encoding into one-hot encoding. The need for this transformation is due to the fact that the inputs of neural networks should be normalized, in order to achieve faster training and remove input bias. In the case of categorical inputs, this can be done by one-hot encoding the feature vector. The resulting vector is a vector with $N \times n_c$ entries, where N is the number of qubits and n_C is the number

$$
\begin{array}{c}
x^1 = \begin{pmatrix} 1 & 3 & 3 \\ 1 & 1 & 3 \\ 2 & 3 & 4 \\ 3 & 1 & 4 \\ 3 & 1 & 1 \end{pmatrix} \\[1em]
x^2 = \begin{pmatrix} 3 & 2 & 3 & 1 & 2 & 1 \\ 3 & 2 & 3 & 1 & 2 & 1 \\ 1 & 2 & 4 & 3 & 1 & 1 \\ 1 & 2 & 4 & 2 & 3 & 1 \\ 1 & 2 & 1 & 3 & 3 & 1 \end{pmatrix} \\[1em]
x^3 = \begin{pmatrix} 2 & 3 & 1 & 2 \\ 2 & 3 & 1 & 2 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 \\ 1 & 1 & 1 & 1 \end{pmatrix}
\end{array}
\xrightarrow[\textbf{padding}]{}
\begin{array}{c}
x^1 = \begin{pmatrix} 1 & 3 & 3 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 1 & 3 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 2 & 3 & 4 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 3 & 1 & 4 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 3 & 1 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} \\[1em]
x^2 = \begin{pmatrix} 3 & 2 & 3 & 1 & 2 & 1 \\ 3 & 2 & 3 & 1 & 2 & 1 \\ 1 & 2 & 4 & 3 & 1 & 1 \\ 1 & 2 & 4 & 2 & 3 & 1 \\ 1 & 2 & 1 & 3 & 3 & 1 \end{pmatrix} \\[1em]
x^3 = \begin{pmatrix} 2 & 3 & 1 & 2 & \mathbf{0} & \mathbf{0} \\ 2 & 3 & 1 & 2 & \mathbf{0} & \mathbf{0} \\ 1 & 1 & 1 & 1 & \mathbf{0} & \mathbf{0} \\ 1 & 1 & 1 & 2 & \mathbf{0} & \mathbf{0} \\ 1 & 1 & 1 & 1 & \mathbf{0} & \mathbf{0} \end{pmatrix}
\end{array}
$$

Figure 3.4: Example of the padding of different encoded circuits. In this case, the 2nd circuit had the most timesteps, and the 1st and 3rd got 0-filled timesteps appended to them, in order to match the number of timesteps of the longest circuit.

of classes per qubit, given by

$$n_c = 1 + N_{single} + \lfloor N/2 \rfloor \tag{3.12}$$

This number of classes (and hence inputs) per qubit is decomposed in: 1 class for idling gates, N_{single} classes for single-qubit gates (1 if only the type is considered, or the number of different single qubit gates if that information is kept), and $\lfloor N/2 \rfloor$ represents the maximum number of concurrent two-qubit gates. An example of this conversion from

$$x_t = \begin{pmatrix} 2 \\ 1 \\ 3 \\ 1 \\ 3 \end{pmatrix}$$

(a) Integer encoding.

$$x_t^T = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

(b) One-hot encoding. The vector was transposed in the one-hot encoding solely for presentation purposes.

Figure 3.5: Example of a quantum circuit timestep, encoded using Integer encoding and One-hot encoding, in a scenario where the information regarding the specific single qubit gate is discarded. It represents the final encoding of the quantum circuit in Fig. 3.2(a), using the encoding rules from Tab. 3.2.

The output data will be constituted by the success probabilities obtained for each circuit via the simulation backend. This data, unlike the input data, needs not be transformed into another representation to be fed to the neural network: it is constituted by purely numerical data and, since it represents circuit success probabilities, it is already in the $[0, 1]$ interval. This approach in which we are trying to predict a numerical quantity frames the metrics problem as a regression problem.

3.2.3 Network Architecture

As aforementioned, RNNs are one of the types of models suited to handle sequential data. This type of neural networks allows the input of sequences of any size (number of timesteps), without the need to change the networks architecture. Furthermore, it also takes into account the order in which the data is fed, whereas simple feed-forward networks allow for no such time encoding. This fact allows it to take into account the circuit's structure when making a prediction.

There are a few variants of RNNs [55, 56]. Some of the most used of these variants are Gated Recurrent Units (GRUs) [57] and Long Short-Term Memories (LSTMs) [58], which try to overcome some of the shortcomings of the original RNNs, such as being able to handle longer sequences and offering increased expressiveness. A simplified overview of the proposed architecture is presented in Fig. 3.6. The architecture is composed of five fundamental layer blocks:

1. **Masking layer:** As explained before, in order for the neural network to be able to process several circuits with different lengths in a single batch, padding is necessary. Masking allows the network to receive the padded data, while minimizing the impact on the network's performance.

2. **Embedding layers:** This block of layers receives the (one-hot) encoded data as input. It allows the network to convert the used encoding into a possibly more efficient one, which the network learns.

3. **Recurrent layers:** This block of layers is responsible for handling the sequential nature of the data. For each iteration through the sequential data, it receives as input the feature vectors encoding each of the quantum circuit's timesteps. The block outputs a single vector: a hidden state that summarises the circuit's information. The fact that recurrent layers take into account the timesteps of each gate, along with the qubit(s) they act on, allows the network take into the circuit's structure, unlike the ESP models. In the performed experiments, Long Short-Term Memory (LSTM) layers were used, which are the most used type of recurrent layer due to its expressiveness. These layers were extended in order to have trainable initial hidden states, to allow for extra expressiveness [59].

4. **Hidden layers:** These layers receive the hidden state given by the recurrent block and outputs another hidden state. While not strictly necessary, this block may improve the networks prediction capability.

5. **Output layer:** Transforms the last hidden state, given by the hidden layers, into our output of interest, a prediction of the circuit's success probability. This layer is composed of a single neuron.

Additionally, Batch Normalization (BN) layers [60] were also added to the model, between blocks as shown in Fig. 3.6, and between layers of the same block as well (with the exception recurrent layer block). Deep Learning is an area of machine learning which sometimes gets plagued with slow learning performance. Batch Normalization is one of the techniques that can oftentimes accelerate training. This procedure aims to improve and speed the training procedure up, by applying a per-layer normalization procedure (represented in Fig. 3.6), based on a rationale similar to the input data normalization.

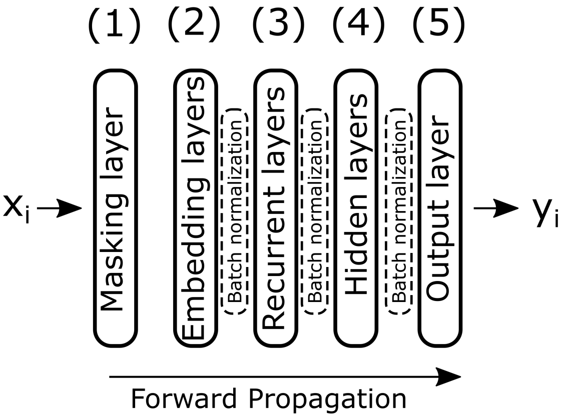


Figure 3.6: Final outline for the neural network architecture used for both the NNESP and NNESP$_{sr}$ models. The neural network receives each encoded circuit (x_i) and outputs the predicted probability of success (y_i). (2) and (4) might be constituted of several layers, depending on the specific hyperparameters used. The dashed BN [60] layers are a training-specific layer meant to speed the training process up, and are not present in the trained network during inference. BN layers are also present between each layer of the layer blocks (2) and (4), but not between the layer blocks in (3), due to a lack of suitability of Batch Normalization (BN) to handle the output of recurrent layers.

In order to train the network, we need a dataset of circuits labelled with their expected probability of success. Since we already have this label for every circuit, and given that it is a single number in the [0,1] interval, no transformations to the label data are required.

3.2.4 Architecture Hyperparameters

Now that the NN architecture outline is defined, there are non-training specific hyperparameters that still need to be fixed. These are:

- **Number of hidden layers**;
- **Number of neurons in each layer**;
- **Activation functions**.

In order to optimize the network's prediction capability in each of the tested scenarios, most hyperparameters were searched after using Random Grid Search [61], as will be described in Sec. 5.2. However, some of the hyperparameters were defined beforehand, as was the case with activation functions. While several activation functions could be used for the majority of the neurons, this is not the case for the output layer neuron. The reason for this is that, the network is expected to output a prediction of a success probability, which is necessarily a number in the [0,1] interval. It is therefore important to bound the output of the network within that interval. In scenarios like this, the Sigmoid activation function is typically chosen, due to having the same bounds. For the remaining layers, the activation of choice was the Hyperbolic Tangent (Tanh). This choice of activation function for the remaining layers comes both from empirical accuracy assessments and training performance and practical reasons:

- **Accuracy:** Some preliminary technical tests were performed in order to choose an activation function. When compared to other commonly used activation functions, like Rectified Linear Unit (ReLU), the usage of Tanh yielded better a better accuracy in the performed tests, for the same network size. A few causes could be hypothesised for this, such as the fact that ReLU is zero in half of its domain, which could otherwise make the network more flexible, or the fact that this activation may also get stuck at zero due to its null gradient for negative domain values. Additionally, the fact that ReLU is not smooth may impact the loss landscape in the same way.
- **Training Performance:** The deep learning framework used for this work, Tensorflow 2.0, possesses GPU accelerated RNNs, which have the ability to perform much faster than normal CPU-run RNNs. These speedups are due to the CuDNNLSTM and CuDNNGRU implementations from the CUDA cuDNN package they rely on. However, these implementations only allow for for the usage of Tanh as activation function.
- **Less hyperparameters to optimize:** As mentioned, in order to optimize the network's prediction capability in each of the tested scenarios, the best hyperparameters were searched after using Random Grid Search [61]. While there were reasons to choose a specific activation only in the recurrent and output layers case, fixing them for the others can also be desirable due to the reduced search space that needs to be scouted.

3.2.5 Training-specific Hyperparameters

Now that the general network hyperparameters are selected, the remaining hyperparameters to be fixed are the training-specific hyperparameters, i.e. the hyperparameters that only present or relevant to the training procedure, but not for the subsequent inference procedures. Among these we count:

- **Optimizer**;
- **Loss function**;
- **Usage of Batch Normalization**;
- **Learning rate**;
- **Batch size**;

- **Number of training epochs and patience**.

These hyperparameters were chosen out of empirical results present in Sec. 5.2.

However, in the particular case of the loss function, some concerns based on previous choices apply. Loss function not suited to the problem at hand may lead to poor learning results and poor training performance. One of the factors leading to a choice of loss function is the type of problem at hand. In this context, one intends the network to predict a numeric success probability obtained from experiments. This makes it not a classification problem but a regression problem. In regression contexts, one typical and intuitive choice of loss function is the Mean Squared Error (MSE). However, another important factor to consider is the output layer's activation function, which is the Sigmoid in the present case. Combining this output activation with MSE is often disadvised, since in this scenario the loss function becomes non-convex [62], which can be problematic when optimizing using gradient descent techniques. An alternative is then to use Binary Crossentropy (BCE), which is typically used in concert with a Sigmoid activation (usually in classification problems, where outputs in the [0,1] range are commonplace). During training, both functions were therefore tested, in order to find the best.

3.2.6 Scalability remarks and possible alternative encodings

The methods described in this present topic are not implemented in this work. However, they can be of interest when considering circuits with a higher qubit number, due to the improved scalability they can offer.

It can easily be seen that the just proposed one-hot representation incurs in a fast growing number of parameters as the number of qubits increases, namely $N_p = N s_e n_c$, where N_p is the number of parameters in the encoding layer, N the number of qubits, and s_e the size of the encoding. This is due to the used encoding allowing two-qubit gates to be performed between any of the devices qubits. However, typically quantum devices have a limited connectivity graph, in which case the present encoding may be wasteful. An alternative encoding can consist in creating just as many classes as needed in order to represent all of the device's links.

An alternative and more scalable encoding (and hence future-proof) may lie in the usage of Graph Neural Networks: these are a recently developed class of neural networks that receive a graph as input, and output an embedding of the said graph. Such type of networks would allow a more efficient encoding of the operation within a timestep, since graphs provide are a natural framework to encode the relationship between the operand qubits of two qubit gates. Hence, it is possible to break free a number of classes defined by 3.12, since there is not longer the need to explicitly encode distinct two-qubit gate with different classes. However, such scalability concerns are left as suggestion for future work.

4

Software Framework and Preliminaries

Contents

In order to evaluate the proposed models, a complete testing framework is necessary. This framework should include: (i) a way to generate a dataset of benchmark circuits; (ii) a backend in order to obtain the reliability data of the benchmark circuits, to be used as ground truth when evaluating the models; (iii) a compiler capable of converting the circuits into code that the backend supports. However, the choice of each of these components is not independent from the others, due to the fact that compilers tend to be tailored to the devices of a specific manufacturer. In addition, the current standard fragmentation in the quantum computing field, typical of a newborn industry, introduces further compatibility constraints between each of these tools. Regarding backends, the diversity of choice is, at the moment, reduced. Two types of backends can be considered: quantum devices and simulators.

Among quantum devices, the only freely and publicly available resources are the IBM Quantum Experience's devices[1]. Despite IBM's system stack and respective Qiskit compiler [65] being relatively mature, there are some factors that render this framework unsuitable for the goals of this thesis. Namely, IBM's system stack implements a final scheduling pass at runtime (to which users don't have access) instead of as part of the compiler. Not knowing the outcome of the last scheduling pass causes a lack of knowledge regarding the structure of each compiled circuit, due to absence of information regarding the time of execution of each gate and the presence of implicit idlying gates.

Conversely, most of the available simulators are noise-free or Pauli Noise Model-based simulators [66, 67], with the notable exception of Quantumsim: a density matrix-based simulation with support for more realistic noise models, which aims to mimic the DiCarlo's Lab superconducting quantum computer implementation [68] via the Qsoverlay configuration layer [69]. Although less ideal than a real device-based backend that is fully transparent, this simulator allows us to know all the information underlying the execution of a quantum circuit, unlike IBM's resources. This makes Quantumsim the preferred backend candidate for the evaluation of the techniques proposed in this work. The DiCarlo's Lab superconducting implementations are capable of the native gate set present in Tab. 4.1.

Subset	Gates
Single-qubit gates	X, Y, RX(45), RX(90), RX(-45), RX(-90), RY(45), RY(90), RY(-45), RY(-90)
Two-qubit gates	CZ

Table 4.1: Gate set containing the primitive gate set for the quantum chips by DiCarlo's Lab. These chips only contain a finite number of single-qubit gates, since they do not currently support RX/RY gates with arbitrary angles. The angles are represented in degrees. CZ is the native two-qubit gate. In these implementations, it takes one cycle (20 ns) to apply single-qubit gates, and two cycles (40 ns) to apply two-qubit gates.

Choosing the Quantumsim simulator as backend made it necessary to use a compatible compiler, known as OpenQL [28], which supports it and the quantum device that Quantumsim aims to mimic.

[1] During the writing of this thesis, a new free service providing access to quantum device backends was made publicly available: Quantum Inspire [63] [64], developed by QuTech. This is a recent service, however, which was not available at the time the experiments in this work were performed.

4.1 Benchmark Circuits

In order to test the predictive performance of the different models, benchmark circuits are needed. They often come in two types: "real" circuits and random circuits. The first represent quantum algorithms used in real-world cases, such as the Ising Model and Quantum Fourier Transform, while the second are circuits created by random sampling of quantum gates.

While random circuits may not be fully representative of the behaviour of real circuits (which tend to have a defined structure), they have the advantage of not being limited in number, since an arbitrary amount can be generated. This ability to generate circuits on-demand is important for the following reasons: (i) it allows tests with statistically significant results to be performed; (ii) it allows reliable studies on the effects of changing depth and number of gates, as well as fraction of two-qubit gates; (iii) a high amount of circuits is necessary to apply machine learning techniques such as the one developed in 3.2. Furthermore, random circuits do not tend to follow any specific structure. This can be advantageous since they allow the unbiased comparison of circuit datasets of different specifications, for example circuit size and circuit depth. Even though publicly available real circuit datasets such as QLib [70] and RevLib [71] are available in the OpenQL-compatible cQASM format, this is not the case for random circuits. To address this issue, a random circuit generator was created, which will now be described.

4.1.1 Random Circuit Generator

Due to the lack of tools capable of generating random circuits compatible with the OpenQL compiler and the Quantumsim simulator, a new random circuit generation procedure was established. It consists of: (i) creating an unscheduled quantum circuit generator which samples from the gate set; (ii) scheduling the generated circuits, using the scheduling procedure implemented in OpenQL.

The unscheduled circuit generator was implemented such that the following circuit parameters could be specified:

- **Number of qubits**;
- **Circuit size (number of gates):** Approximate number of explicit, non-idle gates. The final number may vary due to the use of Initialization, described below;
- **Fraction of two-qubit gates (denoted as f_{2qbg})**;
- **Initialization:** Prepends additional single-qubit gates to the randomly generated circuit. This is an especially important consideration for circuits with $f_{2qbg} \approx 1$ for the following reasons: (i) applying only two-qubit gates like CZ or CNOT on a pool of freshly initialized qubits (each on the $|0\rangle$), will not yield any computation of interest, since the states of the qubits are not changed other than with noise; (ii) phenomena like relaxation make qubits drift to ground state, which coincides with

the ideal output state in this case, effectively masking the noise behaviour and possibly resulting in distorted conclusions; (iii) in most useful computations, qubits are expected to be in some form of superposition, in order to harness some level of quantum speedup. Several initialization scenarios were considered:

  - **No initialization:** Not advised when $f_{2qbg} \approx 1$, due to the reasons mentioned above;
  - **X:** In the first cycle, applying a X gate on each qubit. Likely less than ideal when $f_{2qbg} \approx 1$, due to being very specific and promoting little superposition (other than through noise);
  - **H:** In the first cycle, an H gate is applied on each qubit;
  - **Random:** In the first cycle, random single-qubit gates are applied on each qubit, from the set of gates represented in 4.1;
  - **Random with idle:** In the first cycle, random single-qubit gates are applied on each qubit, including explicit idling gates. Sampled from the single-qubit gate subset (Tab. 4.1) extended with a Idle gate. This was the option used to generated the datasets, due to being the most general.

- **Mirroring:** Whether to generate deterministic circuits, by generating a random circuit and then concatenating it with it's inverse, the latter of which can be computed by inverting the gate sequence and inverting each of the gates – done by replacing them by their transpose conjugate.

The procedure for the generation of each random circuit is as follows:

1. Initialization is performed according to the chosen policy;
2. If Mirroring is activated, loop through the half the number of desired gates, since the mirrored circuit will be appended. Otherwise loop through the number of gates;
   (a) Sample from either the single-qubit or two-qubit gate subsets present in Tab. 4.1, according to the desired fraction of two-qubit gates (f_{2qbg}). Gates within the same set are sampled in an equiprobable manner;
   (b) If the sampled gate is a single-qubit gate, sample on qubit from the qubit pool, otherwise sample two qubits;
   (c) Append the generated gate-qubits pair to the circuit.
3. If mirroring is activated, copy the circuit, invert the copy (by inverting each gate and reversing the order), and append to the original circuit.

When supplied with a set of the parameters above, along with the desired number of circuits to be generated, the program generates the resulting circuit dataset. Each circuit in the dataset is saved in a file with an unique name. A step now remains, namely removing any duplicated circuits which might have been generated.

It is possible that the random number generator produced duplicate circuits. In order to make a reliable and non-biased benchmark set, a procedure that removes those duplicates has to be created.

Directly comparing each circuit with each other may quickly become an insurmountable task, since for a pool of N circuits, $\binom{N}{2}$ circuit comparisons have to be made (each circuit containing itself many gates to be compared). This means $\approx 4.9 \times 10^7$ comparisons for a pool of 10^4 circuits, and $\approx 4.9 \times 10^9$ comparisons for a pool of 10^5 circuits.

A program implementing a scalable method for finding these duplicate circuits was made, using hash tables:

1. In a parallel for-loop, each circuit is loaded from its file and is hashed. Due to its high speed and low number of collisions (defined as equal hashes for different targets), the *sha1* hashing algorithm was used. After this parallel loop, we're left with a dictionary (A), where the keys are the filenames, and the values the respective hashes;

2. Another dictionary (B) is built, with hashes as keys, and the corresponding circuit(s) as values;

3. Dictionary (B) entries with more than a single circuit as value are deleted.

The result is a circuit dataset where every circuit is unique.

4.2 OpenQL Compiler Validation

At the time of writing this thesis, the OpenQL compiler library was still in development, and no validation tests had previously been performed in order to deem the compilation procedure free from bugs. To overcome this issue, a validation procedure (depicted in Fig. 4.1) was created.

One way to verify the integrity of the compiler is to check whether the compiled circuit it outputs results in the same final state as the original uncompiled circuit after simulation, apart from qubit permutations (which may happen if the mapping procedure is run). If the final states of both simulations do indeed match, then it is likely that the compiler procedure only applied computation-preserving transformations. If enough circuits with different characteristics are then used as test subjects, then we can say with confidence that the compiler only performs computation-preserving transformations, subject to specific compilation parameters. The steps that the developed validation tool performs are:

1. Given a set of parameters, arrange a batch of quantum circuits with different characteristics, such that every feature of the compiler is used;

2. Generate the compiled versions of a batch of quantum circuits (given the set of compiler parameters); During this step, the mapping from virtual to real qubits, obtained for each circuit in the compiled batch, is stored;

3. Simulate both the original and compiled batches of circuits without noise and store the results;

4. Check if the results from all simulations are deterministic. That is, if for each circuit, only one output can be measured. If this is not the case for the compiled circuits, then the compiler is classified as faulty (under the specified compilation parameters), and the validation procedure should stop;

5. Apply the mapping obtained for each compiled circuit in step 2, so that the qubit permutations now match. The outputs of any other qubits which might have been used at some point in the computation but not in the compiled circuits case are discarded;

6. Compare the simulation outputs for the circuits of each batch. If there is non-match for the output of any pair of corresponding uncompiled/compiled circuits, then the compiler is deemed buggy.

An overview of the validation procedure can be viewed in Fig. 4.1. It should be noted that, due to possible floating point errors during the simulations, it is possible that the final states, for the same circuit, differ between the compiler's on and off cases by a very small value. A small tolerance margin should therefore be considered when comparing the measurement probabilities of the expected outputs.

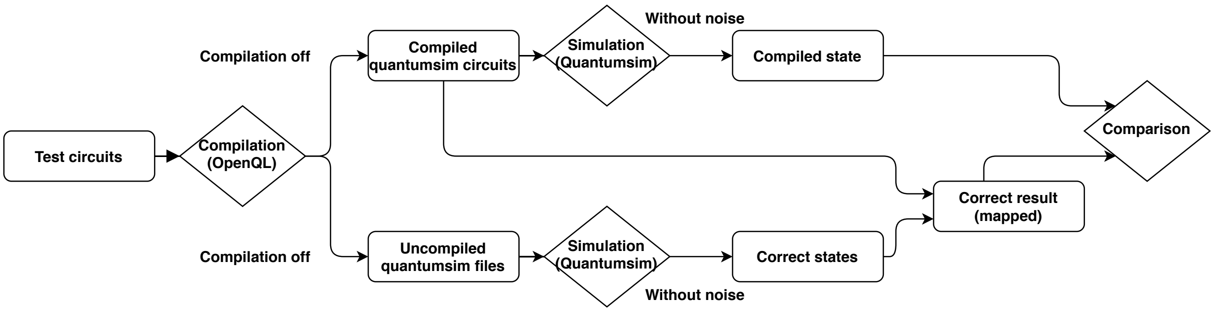


Figure 4.1: Compiler validation procedure overview. Deterministic test circuits are fed into the compiler, which is necessary to generate the file format that Quantumsim receives. For each circuit, a compiled and an uncompiled version of the circuit is generated, in a Quantumsim-compatible format. Both versions of the circuit are simulated without noise, and then compared. Several compilation parameters were tested, including with and without mapping. In the cases where mapping is not used, comparing the final states is trivial. Otherwise, we need to undo the mapping imposed by the compilation procedure on the state. This can be done using a mapping dictionary generated during the mapping procedure.

Using this procedure, a large number of circuits was fed into the compiler and then simulated, and all the occurring compiler bugs were corrected.

4.3 Benchmarks-Compiler-Simulation-Results Workflow

With all of the necessary components already defined, it's time to define the workflow necessary to accomplish the proposed work.

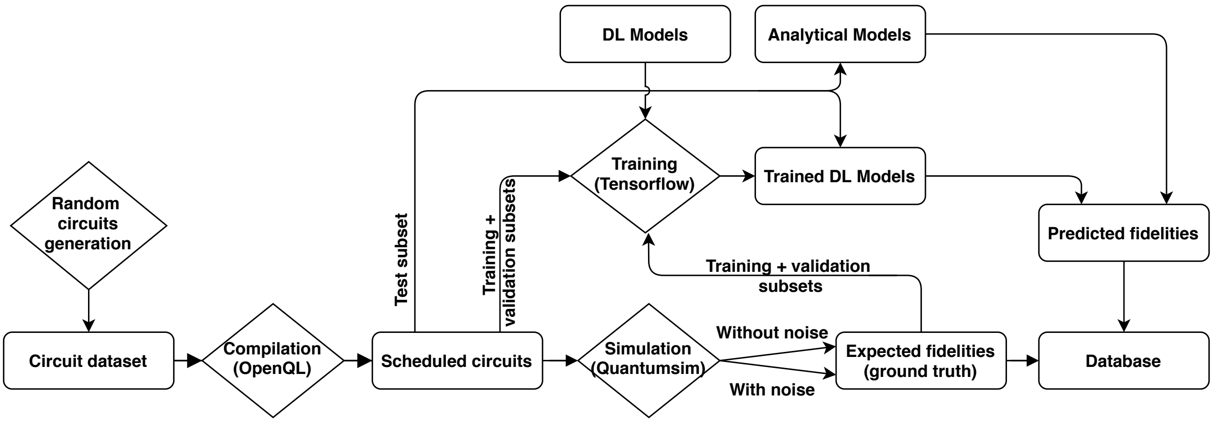


Figure 4.2: Workflow used in order to gather the necessary data to rate and compare the different models, for each dataset. All circuits, besides being simulated with noise, are simulated without it as well, in order to extract the ideal state. This ideal state is then compared against the state resulting from the noisy simulation, in order to extract a success probability for the latter. The dataset of simulated circuits is then split into training, validation, and testing subsets. The training and validation subsets are used in the training procedure for the deep learning models. The testing subset is used in order to generate predictions by every model and used to generate the ground truth against which the predictions will be compared using the methods described in Chap. 5.

As displayed in the scheme in Fig. 4.2, the starting point is a circuit dataset. Different datasets are created (5.1), in order to evaluate the models under different conditions, using the aforementioned Random Circuit Generator. After this, they are fed into a program the circuits are scheduled using OpenQL, in parallel. Since a simulator is used, there is no need to execute the circuits a large number of times in order to extract a probability of success. Instead, the scheduled circuits are simulated using Quantumsim (with a NVIDIA K80 Accelerator GPU) under two different scenarios: one where no noise is present in the computation, and other where noise is also simulated. These two simulations per circuit allow the extraction of a success probability per circuit, by measuring the Trace Distance between the distributions resulting from noisy and noiseless simulation. These success probabilities will then serve as the ground truth against which all models will be trialed.

The scoring of the circuit differs between the analytical and deep learning models, since the latter requires a training procedure as well. For this reason, the circuit dataset should be in training, validation and testing subsets. The first two are used in order to train the deep learning models, which need data in order to learn how to predict. The testing subset is fed to all the models (both analytical and trained deep learning models), in order for its circuits to be scored according to them. The training and evaluation procedures for the deep learning models were performed using the popular Tensorflow [72] deep learning framework, using NVIDIA K80 Accelerator GPUs. Finally all this acquired data is stored in a database, implemented using the *pandas* python package, which is then analysed in Sec. 5.

Due to the complexity of performing a through GST procedure, the fidelities for each gate were not averaged for all states, but were instead extracted from the state yielding the worst case scenario. The considered fidelities for each gate are displayed in Tab. 4.2.

Gate	Fidelity
I	0.99910
X	0.99941
Y	0.99941
RX/RY(±45)	0.99970
RX/RY(±90)	0.99958
CZ	0.99680

Table 4.2: Gate fidelities extracted from the DiCarlo's superconducting chip setup in Quantumsim, corresponding to the worst case.

5

Results and Model Comparison

Contents

5.1 Testing Methodology

More often than not, articles proposing metrics for the mapping problem tend to evaluate them in a algorithm-dependent fashion. As mentioned, this can result in an evaluation which can be biased by the ability (or lack thereof) of the algorithm to navigate that metric appropriately. Hence, this work proposes a new method to evaluate a given metric.

One of the major roadblocks in accomplishing an algorithm-independent evaluation of metrics is that not all metrics express the same quantity, they just output quantities which aim to correlate with the success probability. Examples of these are circuit size and depth, which are simply quantities negatively correlated with success probability. ESP, on the other hand, despite being a probability and correlated with success probability, tends to diverge from the latter as circuits become larger. In order to compare them, a possible strategy is to look for monotonic correlations between the each of the metrics and the simulated success probability (ground truth). This can be done by ranking the circuits using a pointwise ranking method [73]: it is possible, for each metric, to construct a circuit ranking via the score attributed to each circuit. This then allows the comparison of the different metrics via the comparison of the induced circuit rankings. The resulting rankings can be compared with the ground truth ranking, induced by the obtained success probability, in order to extract a measure of correlation. Specifically, the proposed metric is described by the following steps:

1. Score a set of test circuits (random circuits, for example) using the various metrics to be compared, including the simulated success probabilities;

2. Rank the circuits according to the scores obtained for each metric, the end result being a ranking for each of the metrics;

3. Check which rankings are the closest to the one induced by the simulated success probabilities. This can be quantified using a measure of distance between rankings or rank correlation.

Examples of popular measures of rank distance are the Kendall Tau Distance and Spearman's Footrule, which are associated with the correlation measures they respectively power, Kendall's [74, 75] and Spearman's [76] rank correlation coefficients. Due to it's robustness to outliers, the Kendall set of methods was chosen, specifically its correlation coefficient, since the latter allows the comparison of ranking accuracy between sets with different sizes. The Kendall τ coefficient measures the similarity between two rankings, by verifying the number of order inversions between any two elements, from one ranking to the other. The τ coefficient between two rankings, A and B, is given by:

$$\tau(A,B) = 1 - \frac{2D_{A,B}}{N_p} = 1 - \frac{2D_{A,B}}{\binom{N}{2}} \tag{5.1}$$

where $D_{A,B}$ is the number of discordant pairs (pairs of elements appearing in opposite order in each ranking), and $N_p = \binom{N}{2}$ is the total number of pairs possible between any of the ranking's elements, with N being the number of elements in each ranking. A Kendall coefficient of $\tau = 1$ means that two rankings are perfectly correlated (i.e. are equal), $\tau = -1$ represents perfectly negatively correlated circuits (one is the reverse of the other) and $\tau = 0$ denotes no correlation between the rankings. An illustrative example of the usage of the Kendall τ coefficient is portrayed in Tab. 5.1.

5.1.

Rank	Model A ranking	Model B ranking
1	circ1	circ3
2	circ2	circ2
3	circ3	circ1
4	circ4	circ4

Table 5.1: Example of two dummy rankings, that could be generated by two metrics, for a set of four circuits (circ1-4). The Kendall Tau coefficient between these rankings would be evaluated by calculating the number of pair permutations (discordant pairs) occurring between the two rankings: 3 pairs (circ1,circ2), (circ1,circ3) and (circ2,circ3) are permuted between the two rankings, i.e. $\tau(A, B) = 1 - \frac{2\times3}{\binom{4}{2}} = 0$. This means that A and B are not correlated.

Using this described method, it is possible to evaluate how good a metric is. Given a dataset with several quantum circuits, rankings are generated by both the candidate metric and the simulated success probability, with the latter acting as ground truth. Afterwards, the Kendall the τ correlation coefficient between the two rankings is calculated, and is used to estimate how good the candidate metric is. A value of $\tau = 1$ would mean the metric is perfect in the context of the given dataset.

In order to generate reliable results, a big and diverse dataset is required. To achieve this, two datasets with a variety of parameters were generated, using the methods described in Sec. 4.1.1 and the parameters in Tab. 5.2.

Name	Nr. qubits	Size (Min, Max, Step)	Two-qubit gate fraction	Samples per spec
A	5	(0, 50, 1)	{0, 0.2, 0.4, 0.6, 0.8, 1.0}	100
B	5	(0, 50, 1)	{0, 0.2, 0.4, 0.6, 0.8, 1.0}	1000

Table 5.2: Circuit datasets generated using the Random Circuit Generator described in Sec. 4.1.1. The Samples per spec column displays the number of generated circuits for each of the configurations portrayed in the previous columns. Both datasets were generated using the ”Random with idle” initialization, and no ”Mirroring”.

By splitting the generated datasets, present in Tab. 5.2, more datasets were created (Tab. 5.3), which will allow us to evaluate the models in different scenarios: (i) Varying amounts of training data, allowing us to recognize the impact of this on the evaluation accuracy; (ii) Varying depth, aiming to evaluate the usefulness of the predictions for mappers using different look-ahead/look-back windows (check the Compartmentalization technique in Sec. 2.5.1). Since these windows are typically no longer than 50 cycles, this was the chosen value was the maximum circuit depth considered for the datasets.

Parent Dataset	Child Dataset	Max Depth	Training Samples	Total samples
A	A1	50	20373	29098
	A2	15	10070	14384
B	B1	50	291150	203895
	B2	15	100212	143342

Table 5.3: Final circuit datasets used to compare the models and train the deep learning model. Each dataset was split into training (70%), validation (15%) and test (15%) datasets. The training and validation datasets were used solely for the training procedure for the deep learning models, while the test datasets were used for the evaluation and comparison of the different datasets, through usage of the Kendall τ correlation.

The final circuit datasets presented in Tab. 5.3 were the ones using for the tests made in the following sections.

5.2 Deep Learning Models: Training and Tuning

Developing a good deep learning model is often an iterative process. This is partly because neural networks behave, in large part, as black boxes, from which it is hard to withdraw conclusions on a per-component basis. There are several knobs that have to be adjusted, called hyperparameters, that are not learned by the model itself, and typically can not undergo gradient descent-based optimization.

In order to come up with a good selection of architectural (number of layers/neurons per layer) and training-specific hyperparameters (batch size, etc.), the problem was split into several parts, in order to reduce the optimization complexity. In an initial phase, the best training-specific hyperparameters were sought, while the architectural hyperparameters were fixed according to an educated guess: a four layer NN was used (one initial feed-forward layer, a single LSTM layer, followed by another single feed-forward layer, and the final output layer), with 5 neurons per layer (except for the output layer, where a single neuron was used). The rationale behind this ansatz was to attempt a NN model with dimensions similar to the quantum circuits that are to be received, which are 5-qubits wide in the generated datasets.

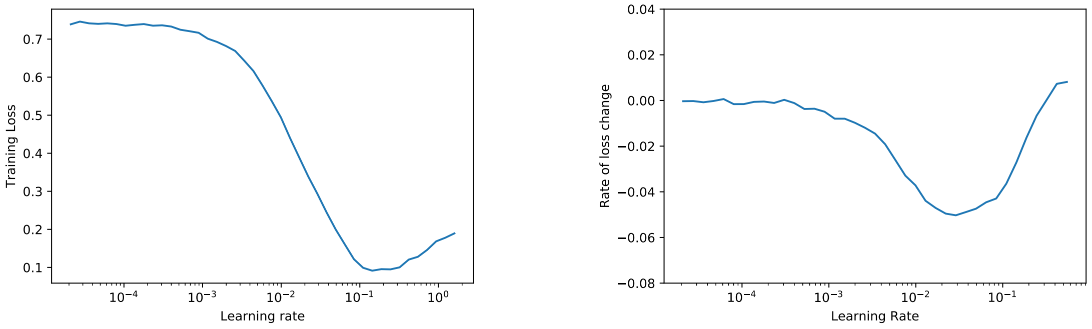
Afterwards, the best training-specific hyperparameters found (except for the learning rate, as will be explained) were fixed and used for the rest of the work, including during the search for the best architectural hyperparameters.

5.2.1 Training-specific Hyperparameter Selection

In order to drive the loss function down, by adjustment of the parameters in the best way, the usage of an optimizer is required.

A variety of optimizers is available, each with different strengths, such as accuracy or training speed [77–79]. Examples of popular optimizers are Adam [80], Stochastic Gradient Descent (SGD) [81, 82], and variants thereof. Due to its good combination of training speed and accuracy, the Adam algorithm was chosen [79].

Given a certain optimizer, the batch size and the learning rate cannot be chosen independently. Too small a learning rate for a certain learning rate is likely to result in slow convergence. Conversely, if the learning rate is too large, then the optimizer will make training process unstable. While optimizers will usually dynamically adapt the learning rate during training, they still require an initial value. A good approximation for this initial value is sometimes obtained in a grid-search fashion. However, in order to efficiently compute an initial learning rate value for every instance of hyperparameters and network architecture, the technique introduced in Smith et al. [83] and implemented in [84] was used. This allows us to gauge the performance of each candidate value, by visualizing the achieved loss for each learning rate – see Fig. 5.1(a). The candidate corresponding to the most negative derivative of the loss (that is, with the fastest decrease in loss), corresponding to the minimum in Fig. 5.1(b), is selected as the optimal initial learning rate.



(a) Loss plot.

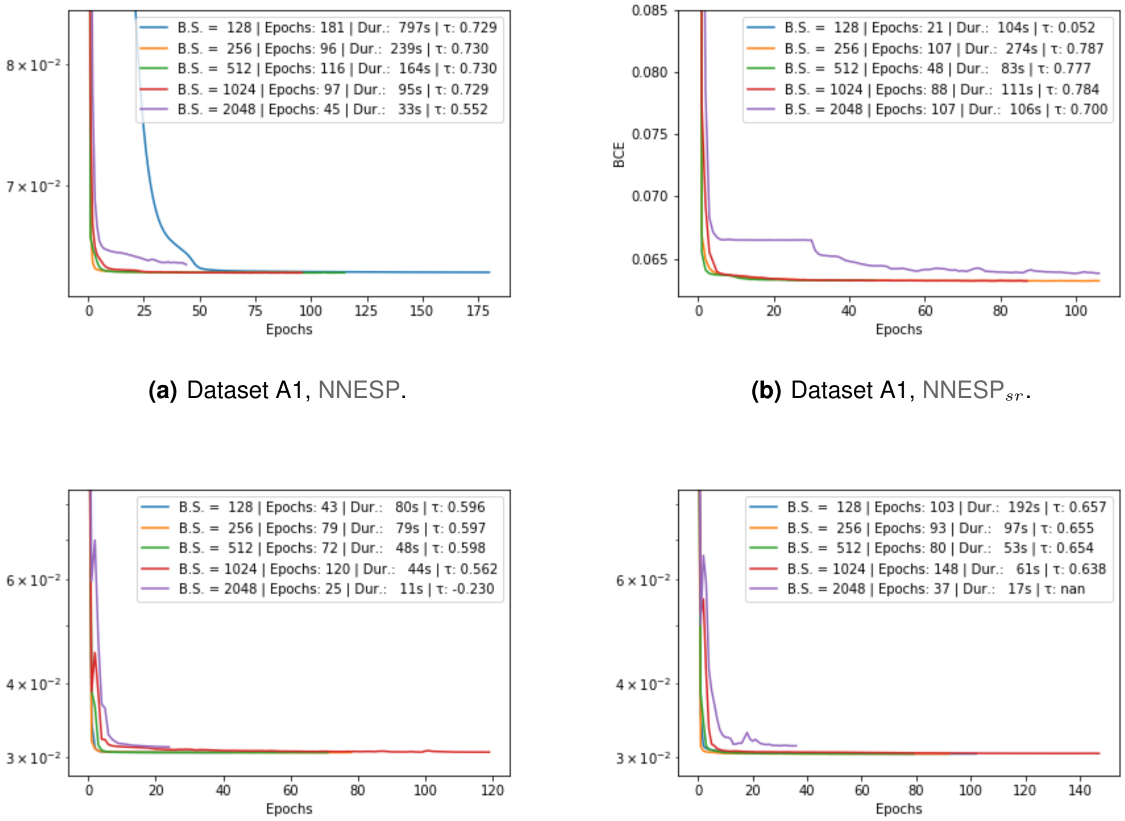
(b) Loss change plot.

Figure 5.1: Search for the optimal learning rate, using the method introduced by [83]. The loss change plot (5.1(b)) was smoothed using the Simple Moving Average (SMA) considering, for each timestep, the previous 4.

Using the above described technique to calculate the learning rate, grid-based searches were performed with varying batch sizes, both loss functions, and both with and without Batch Normalization (in order to confirm it produced the desired effects), for the A1 and A2 datasets (since they are smaller, but have otherwise similar features to the B1/B2 datasets), using both the lossy (NNESP) and lossless (NNESP$_{sr}$) variants of the deep learning approach.

In these tests, the BCE loss scored an advantage in the majority of the dataset/NN variant/hyper-parameter combinations, when compared to MSE, irrespective of the dataset and NN variant used: globally, without batch normalization, it yielded a better Kendall τ correlation on 16/20 performed tests, while with batch normalization it excelled in 17/20 tests. As for batch normalization itself, while it did produce a negative correlation variation of less than 1% on most tests, it sped the training times up by an average factor of $3.5\times$. For this reason, the choice was to use both Batch Normalization and BCE (in detriment to MSE) in the subsequent training procedures.

From this grid-search, it was also concluded that varying the batch size did not introduce any visible tendencies in the resulting Kendall τ correlation. Furthermore, it was also confirmed that the training could be comfortably accommodated under a maximum number of 2000 epochs and, as such, this value was maintained for further experiments. As for the Patience value it was set at 20 epochs, after empirical trials. This value allows the early stopping of a training procedure in case the validation loss stops dropping during this number of consecutive epochs.



(a) Dataset A1, NNESP.

(b) Dataset A1, NNESP$_{sr}$.

(c) Dataset A2, NNESP.

(d) Dataset A2, NNESP$_{sr}$.

Figure 5.2: Training loss (in the form of Binary Crossentropy (BCE)) for the A1 and A2 datasets, using both the lossy (NNESP) and lossless (NN$_{sr}$) deep learning models. For each of these configurations, various batch sizes were experimented with. Most training runs ended well before the maximum number of 2000 epochs, due to the Early Stopping Mechanism, which stops training if the validation loss stops decreasing for a set number of epochs (patience), 20 in this case. In these test sets, Batch Normalization was used.

From the plots in Fig. 5.2, it can be seen that the batch size appeared to have little to no effect on the obtained Kendall τ correlation, as no clear correlation between these two values can be seen. There were, however, some outliers. Namely, there were some instances where a given batch size appeared to give origin to disparate τ values: this incident was particularly frequent for batch sizes of 2048 samples, the cause for which was not ascertained. Due to the otherwise poor correlation between batch size and τ values, and due to the said incidents, most tests relative to the following sections were run using a batch size of 1024 sample, lowering this amount to 512 samples in any further outliers were detected. A large batch size was preferred since, despite having little impact on the correlation, larger batch sizes are able to provide faster training procedures, due to the more efficient parallel utilization of resources.

5.2.2 NNESP Architecture Hyperparameter Tuning

Once the best training-specific hyperparameters for the the developed deep learning model were estimated, several tests were performed in order to find the best remaining hyperparameters, namely number of neurons per layer and number of layers. This is relevant, since too small a network may incur in underfitting (can't fit the training data), while too big a network may cause overfitting (tries to learn too much from the training data, which leads to poor generalization when applied to other set of data). Furthermore, it is of interest to find how much we can reduce the architecture size in order to improve its scalability (in the form of minimizing network size for a given number of qubits) and performance (so that training and evaluation are fast).

In order to find the ideal network configuration in the sense of Kendall correlation, a search space was defined, as present in Fig. 5.3, in order to find the model configuration yielding the best Kendall correlation (τ optimized). In order to explore this combinatorial search space in a feasible amount of time, Random Grid Search was used [61]. The results are displayed in Tab. 5.4.

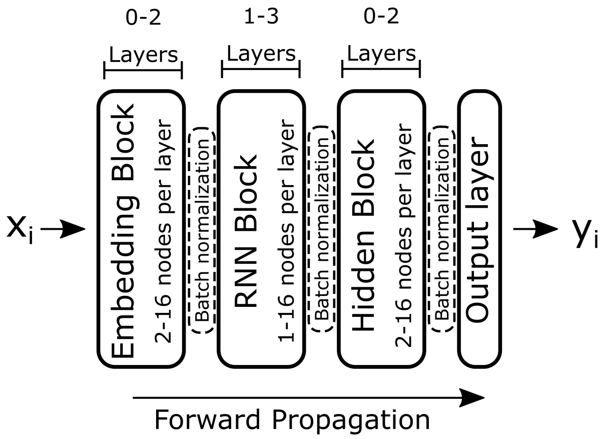


Figure 5.3: Hyperparameter search space defined during the optimization of the neural network's architecture, namely number of layers per block and number of nodes per layer. The output layer, consisting of a single output neuron, was left untouched. As before, Batch Normalization layers were inserted between every pair of layers (except within the Recurrent layers block).

Model	Dataset	Embedding block	Recurrent Block	Hidden Block
A1	NNf	[12,0]	[3,0,0]	[7,0]
	NNf$_{sr}$	[16, 2]	[16,16,1]	[2,10]
A2	NNf	[2, 8]	[6,7,1]	[4,8]
	NN$_{sr}$	[6, 6]	[10,11,1]	[15,7]
B1	NNf	[10, 12]	[12,7,14]	[10,7]
	NNf$_{sr}$	[16, 2]	[11,12,12]	[11,13]
B2	NNf	[6, 2]	[11,7,4]	[8,14]
	NNf$_{sr}$	[4, 12]	[12,6,12]	[10,8]

Table 5.4: Results from the τ optimization step using Random Grid Search.

It should be noted, however, that the models resulting from the τ optimization are likely oversized, since no soft bounds were set for the model size. We demonstrate this in Tab. 5.5, by comparing the τ optimized architecture for the A1-NNESP dataset-model combination with a smaller, manually found architecture yielding similar results.

Configuration	Embedding block	Recurrent Block	Hidden Block	Val. Loss (BCE)	Kendall τ
τ opt.	[12,0]	[3,0,0]	[7,0]	0.06391	0.7308
Size opt.	[2,0]	[3,0,0]	[0,0]	0.06392	0.7291

Table 5.5: Results for the τ optimization step (τ opt., using Random Grid Search). Afterwards, the smallest model yielding a τ value within a 0.02 interval was found (Size opt.). The number of neurons per each of the layers within each block is displayed.

The global results displayed in the following section were obtained using networks optimized solely for Kendall correlation, but not for network size.

5.3 Metrics Comparison

Through the usage of the datasets presented in Tab. 5.2, it was possible to rate the ranking prediction accuracy of each of the presented metrics models. The results are displayed in Tab. 5.6.

The best faring analytical models were the variants of the Estimated Success Probability, proposed by Nishio et al. [36]. These models were able to beat the other experimental analytical models (TESP1, TESP2) by a reasonable margin.

The first conclusion that can be drawn from the several ESP implementations are that taking the idling gates into consideration did improve the predictive power of the model, albeit marginally. However, taking the specific gates' errors into account, rather than a generalized per-gate-type error, did not yield improvements, actually resulting in an accuracy reduction.

The deep learning models, based on supervised neural networks, did indeed outperform the competing analytical models in all cases. The relative accuracy improvements, as compared to the best analytical model (ESP^{i}), ranged from minimal improvements of around 2%-5% when considering only

		Kendall τ Correlation				Improvement relative to ESP^0			
		A1	A2	B1	B2	A1	A2	B1	B2
Metric	Circ. size	0.537	0.446	0.541	0.457	0.75	0.78	0.76	0.78
	Circ. depth	0.727	0.595	0.725	0.605	1.02	1.04	1.01	1.04
	N$_{2qbg}$	0.634	0.313	0.637	0.324	0.88	0.55	0.89	0.55
	TESP1	0.630	0.409	0.471	0.416	0.88	0.71	0.66	0.71
	TESP2	0.519	0.426	0.522	0.433	0.72	0.74	0.73	0.74
	ESP^0	0.716	0.573	0.714	0.584	1.00	1.00	1.00	1.00
	ESP^i	0.719	0.582	0.717	0.591	1.00	1.02	1.00	1.01
	ESP$^0_{sr}$	0.709	0.572	0.707	0.576	0.99	1.00	0.99	0.99
	ESP$^i_{sr}$	0.697	0.522	0.695	0.525	0.97	0.91	0.97	0.90
	NNESP	0.731	0.602	0.731	0.611	1.02	1.05	1.02	1.05
	NNESP$_{sr}$	$\boxed{0.783}$	$\boxed{0.657}$	$\boxed{0.817}$	$\boxed{0.696}$	$\boxed{1.10}$	$\boxed{1.13}$	$\boxed{1.14}$	$\boxed{1.18}$

Table 5.6: Ability of each model to correctly rank each circuit datasets. The Kendall τ correlation between each model and the ground truth is presented for every model-dataset combination (for all models: A1, A2, B1, B2). Additionally the normalized correlations relative to the ESP^0 model implemented in Nishio et al. [36] are also displayed. N$_{2qbg}$ is the identifier for the two-qubit gate metric. In the case of the NN-based models, the displayed correlation values were obtained after τ-driven hyperparameter optimization. The boxed values represent the best correlation obtained within each dataset.

gate types, to significant improvements of 10%-18% when distinguishing between single-qubit gates. Regarding the training process, it can be seen that, while not much improvement resulted from a larger dataset in the gate type-only case, it did make a difference in the full-information case.

However, it's hard to assume how the results would translate when considering real, quantum device-based experiments: these have not only more complex noise properties, but also varying fidelity values between qubits and links (used for two-qubit gates). These may still significantly impact the quantum computation, which are not replicable in simulations with the current state-of-the-art of noise models.

Another important point to take into consideration is the circuit evaluation time. This is relevant, since these methods should be fast in order to be suitable as a mapping algorithm metric, and one of the reasons simulation is not a good procedure to obtain a metric (even more so when dealing with density matrix-based simulators).

Average circuit scoring time (10^{-4}s)							
Dataset	Simulator	ESP	ESP^i	ESP$_{sr}$	ESP$^i_{sr}$	NNESP	NNESP$_{sr}$
A1	57.94	0.22	0.44	0.17	0.19	8.31	8.45
A2	36.60	0.13	0.15	0.12	1.42	5.38	6.09

Table 5.7: Average evaluation time per circuit (in 10^{-4}s, correspondent to the (Quantumsim) simulator and the best faring models from Tab. 5.6. These times were extracted from the A1 and A2 datasets, containing up to 50 and 15 cycles, respectively, since average evaluation times are not expected to vary for the big datasets (B1, B2), which contain circuits sampled the same distribution. It should be noted that the NNESP and NNESP$_{sr}$ times include the procedure of transforming the quantum circuits into the one-hot representation feedable into the neural networks. All operations were computed using Python, with the evaluation running on the Nvidia Tesla K80 GPU for the Quantumsim simulation and neural network evaluation.

As expected, the average circuit evaluation time is higher for the deep learning-based models, since the process of encoding the circuit into the NNs' input representation and further evaluation is time consuming. This contrasts to the lighter computation required for the ESP models, which boil down to counting the number of gates of each type/variant in each circuit and executing the required formula (Eq. 2.6). The time taken by the simulator, however, is already one order of magnitude higher than the neural network, for the current amount of 5 qubits, and it is expected to climb sharply, which quickly rules it out as a way to estimate success probabilities.

5.4 Absolute Circuit Scoring

As argued, the circuit ranking correlation method was used as a way to compare the different circuit scoring models, since the values they output are not directly comparable: the values resulting from the models represent different mathematical quantities. Circuit rankings can help during compilation processes such as routing, however they can't tell the user the absolute success probability of a circuit. However, since the developed deep learning models not only try to correlate with the circuits' success probabilities, but actually try to mimic their absolute value, they may become useful for new use cases which might require such a value.

In order to visualize how closely the deep learning models can mimic the success probability, Fig. 5.4 is presented comparing, for the B1 dataset, the real success probabilities and those predicted by the best analytical and deep learning models (ESP$^i_{sr}$ and NNESP$_{sr}$, respectively). The B1 dataset was used due to its higher number of samples, allowing the data-driven NNESP$_{sr}$ model to achieve the best possible accuracy, and due to its higher maximum circuit depth, allowing its circuits to display a wider range of success probabilities. Tab. 5.8 quantizes the regression quality for the two models.

Model	MAE	Avg. diff. (10^{-2})	R^2
ESP^i	0.0638	-5.97	-45
NNESP$_{sr}$	0.0025	0.00691	0.85

Table 5.8: Regression quality of the best analytical model (ESP^i) and the best deep learning model (NNESP$_{sr}$). Displayed are the Mean Absolute Error (MAE), the Average Difference, and R^2.

It can be see from Fig. 5.4 and Tab. 5.8 that the NNESP$_{sr}$ model is a considerably better estimator for the success probability, yielding very close values in most of the considered domain. Comparatively, the ESP^i model does not output a similar value, and tends to further diverge as the real success probability (as computed by the Quantumsim simulator [68]) drops. Furthermore, the ESP^i model presents a much larger dispersion of results, as can be seen by the width of the percentile bands. As can be seen in Tab. 5.8, the MAE tells us that the NNESP$_{sr}$ model's error is on average more than one order of magnitude smaller than the ESP^i's error. Furthermore, the absolute value of the average difference is more than three orders of magnitude higher in the ESP^i case, expressing a much larger bias that its counterpart.

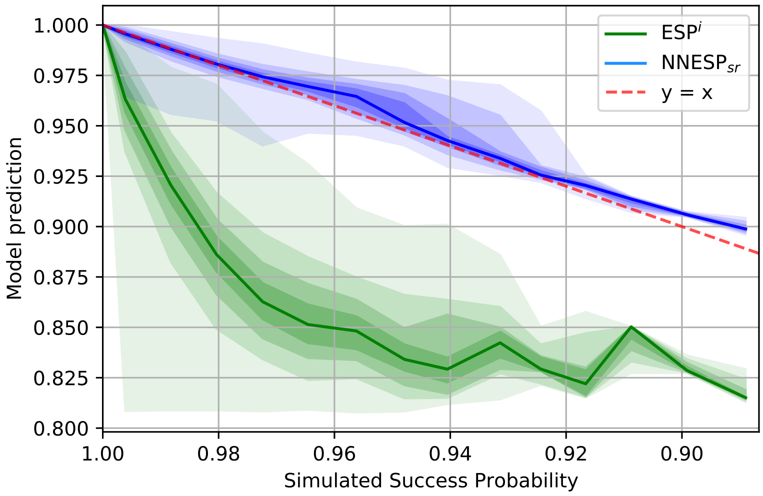


Figure 5.4: Demonstration of the deep learning models' capability to mimic the real success probability values. We compare the predictions from the deep learning and analytical models which best correlate with the success probability – NNESP$_{sr}$ and ESP^i respectively – in the context of the B1 dataset. The 25%, 50%, 75% and 100% percentile bands are displayed. The minimum displayed real success probability value is ≈ 0.88, due to the dataset's maximum depth of 50 cycles, which are insufficient for the success probability to drop further than this.

The fact that it is negative means that ESP^i tends to underestimate the real success probability value, a fact that is not surprising due to the simple recurrent gate fidelity multiplication method it relies on, which tends to quickly converge to zero. Finally, the R^2 tells us that how well the models fit the data compared to the constant model yielding the average real success probability for every circuit. The calculated values reinforce the suspicion of NNESP$_{sr}$ being a good estimator of the success probability. On the other hand, ESP^i's negative R^2 value reflects the fact that it actually performed worse as a estimator than the constant model. The estimation capacity of the deep learning models didn't seem to hold for circuits longer than 50 cycles, however: the network appears to hit a plateau and attribute a success probability of ≈ 0.88 to such circuits. This is likely due to the fact it was not trained on circuits with success probabilities lower than this.

Additionally, Fig. 5.4 depicts the fact that the dataset's samples may not be regularly distributed across the presented success probability interval, which might contribute to model bias. We suggest, for future works, researching how to best construct datasets in order to make the most out of data-driven models.

These novel deep learning models are, as far as the author knows, the first capable of reasonably estimating the success probability of a short circuit, in a single-shot evaluation, without requiring complex and costly simulations or running the circuit in a quantum device a high number of times – usually in the order of thousands, to reduce statistical errors.

6

Conclusions

Contents

6.1 Summary

The goal of this work was to attempt to find the best way to predict the relative success probability of different quantum circuits, as motivated by the needs present in the Qubit Routing problem. Given that a real device with the presented requirements was not available, the tests were performed in a density matrix-based simulator called Quantumsim, using the experimental parameters extracted from a real device by the DiCarlo Lab.

The model evaluation/comparison problem was stated: although correlated with success probability, the majority of the models outputs different mathematical quantities, which cannot be directly compared. In order to solve this, a novel method was proposed: one that uses each model to score a set of circuits, and rank them accordingly; afterwards, a ranking correlation method (such as the Kendall τ correlation factor) can be used to calculate the correlation with the true circuit ordering (obtained via simulation). This new method allowed the unbiased comparison of several existing models, as well as the newly developed ones.

Using this method, various models previously used in literature were tested, namely circuit size, circuit depth, number of two-qubit gates and Estimated Success Probability (ESP). New models, both of analytical an deep learning nature were also proposed and evaluated. The proposed analytical models (TESP) aimed, unlike all the previous analytical models, to take the structure of quantum circuits into account by modeling reliability as a decoupled, per-qubit entity, affected by gates. The deep learning models, on the other hand, aimed to take a data-driven approach at solving the metrics problem, due to the complexity of quantum noise behaviour, and the difficulty and cost in developing analytical models ensured by this complexity.

It was found that, within the analytical models, the ESP-based models were the best performing. Among these ESP-based models, the best results were obtained by considering only the gate-type of each gate (i.e. discarding the information regarding the specific rotation) and considering implicit idling gates. Among the remaining analytical metrics, circuit depth performed the best. One should keep in mind, however, that this might not be the case for every device, but rather possibly the effect of the balance between qubit lifetime errors and non-idle gate-induced errors on this specific backend. As such, in this case, this result might indicate that qubit lifetimes might be the primary source of error on this backend. It is possible that, on devices whose error rates are mainly driven by gate quality, that depth may fall short of other metrics, such as size. The proposed TESP models, however, only managed to fare comparably to circuit size, depth and two-qubit, which indicates that further refinement might be needed.

All the deep learning-based models, on the hand, performed consistently better than any of the analytical models, accuracy-wise. In particular, the NNESP$_{sr}$ model, discarding no information regarding single-qubit rotations, yielded improvements relative to the best analytical model (ESP^{i}) in the 10%-18%

range. It was found that, in the deep learning case, discarding no single-qubit rotation information was indeed highly beneficial, contrary to what was witnessed in the ESP-based counterparts.

Besides displaying the best accuracy (although at the cost of higher evaluation speed), the proposed deep learning models also exhibited a previously unseen perk: the capacity to mimic the absolute values of success probability in shallow circuits (up to 50 gates). This may allow new use cases where such quantities are of interest (without the need to simulate/run the circuits in quantum devices and perform sampling). As an added advantage, the deep learning model does not need parameters such as gate fidelities and coherence times to be extracted from the device, like the other models do.

Additionally, an important advantage of data-driven models such as those based on machine learning (especially deep learning), is the ability to take into to account the specific dynamics of different devices: each technology has a different noise behaviour, which can be hard to model, and might require different analytical success probability models for different devices/technologies. Additionally, noise behaviours vary even between devices based on the same technology. Ultimately, machine learning models also have the ability to learn about complex noisy behaviours which are hard to model, such as *leakage* [85] and *crosstalk* [86]. By using data from the device itself, constructing a working model may prove itself easier and less costly compared to alternative, purely analytical methods.

6.2 Future Work

Deep learning based-models still face a couple of hurdles. Although lower evaluation speed, when compared to analytical models, may be compensated by increased accuracy, the qubit scalability problem needs to be tackled: the current model requires a growing number of parameters, as the number of qubits and the connections between them grows, and more input nodes become therefore needed. One possible approach to tackle this issue is to change the way circuits are encoded and inputted to the network, by using Graph Neural Networks (GNNs) [87]. This type of networks take graphs as input (instead of vectors): this can prove beneficial since graphs are the natural way to represent a set of interactions between a number of qubits and other data of combinatorial nature. Additionally, some new techniques may be useful, in order to allow networks to learn some concepts more effectively with data, while reducing the number of trainable parameters. As an example, it is not efficient for a network to have to relearn the concept of a two-qubit gate combination of two qubit gates, or link: instead, sharing this learned information may improve the training process, yielding lower training times and higher accuracy, reduce the data requirement and boost scalability. Such conceptual invariance may be encoded via the usage of convolutional approaches, such as the ones that constitute the basis of Convolutional Neural Networks (CNNs) [88].

Another scalability-related avenue is scalability with circuit depth, providing the ability for longer circuits to be used during training and evaluation time. This can be interesting, since future qubit routing algorithms may be able to use bigger windows (bigger than the 50 cycles that were considered in this work), in order to optimize the mapping problem in a less local manner. However, the usage of RNNs as a means to handle the sequential nature of the data may prove to be less than ideal for such longer depths, since training and evaluation times can become very long and memory usage too large. In fact, RNN training procedures using too long circuits may not be effective at all, since long Backpropagation Through Time (BPTT) routines may result in vanishing gradients when adjusting the network's parameters. A way to tackle this may lie in the use of newer sequential models, such as Transformers [89] or Reformers [90], which tend to solve similar speed- and sequence length-related issues. The capacity to successfully predict the success probability of arbitrarily deep circuits would allow one to know beforehand whether a quantum device is able to run a whole quantum program with a given reliability. Additionally, such a capable model could also serve as a benchmark for quantum devices, and reduce the need for other more abstract benchmarks such as Quantum Volume [91].

It is also of interest to develop a good method to encode arbitrary gates. As should be reminded, in this work only a finite single-qubit gate set was considered (a number of predefined rotations along the various axes). While these are the only gates allowed by the quantum device that Quantumsim tries to mimic, the tendency is for devices to allow arbitrary single qubit gates, by allowing arbitrary angular rotations. Harboring this infinite gate set will require a very different approach for single-qubit gates encoding. One possible way to solve this might be to encode single qubit gates as a axis-angle pair of parameters: for example, encoding categorically encoding the axis (as one of 3 axis classes), while encoding the angle as a fractional, normalized number.

A third goal that is ultimately necessary to indicate overall feasibility of deep learning models, is to study of the behaviour of the presented deep learning techniques in a real-world setting, i.e. a real quantum device, instead of just relying on simulations. This is important, since real devices have more complex noise behaviours than what is possible to replicate in state-of-the-art simulators. This raises the question of how well the presented models can be applied in such settings. Answering this question will require the investigation of efficient ways to extract success probabilities to serve as labeling data, and the addition of measurement operations to the models. Addition of measurement operations to the proposed models should be straightforward, since they can be encoded in a way similar to single qubit gates.

Additionally, the creation of datasets for the deep learning models can itself be subject of study. The quality of a deep learning project not only depends on the neural network's architecture and on other hyperparameters, but also on the quality of the dataset from which it is expected to learn the intended behaviour. Such study should answer the question of how to structure a good dataset, such that the trained network is able to perform well regardless of the characteristics of the circuit and the fraction of qubits that are effectively used.

Other avenues of research are also possible within the machine learning realm, both inside and outside the deep learning framework. On one hand, inside the deep learning framework, other Machine-Learned Ranking approaches [73] can be pursued, other than the current Pointwise approach, namely the Pairwise (learning to rank, by learning to perform binary comparisons between circuits) and Listwise (learning to rank by considering the entire set of circuits) approaches [73]. Furthermore, fast deep learning-based quantum circuit simulation [92] may also prove itself useful on this topic. On the other hand, outside the deep learning framework, other noteworthy methods should be mentioned. Examples of such methods are decision tree based algorithms [93] (such as Random Forests and Gradient Boosting) and Genetic Programming [94]. Although it may be argued that these methods have less potential for accuracy that deep learning [95], the increased explainability they offer may make for easier scalability. Genetic programming, in particular, may be especially interesting, since it is a data-driven method that can find analytical expressions that model a specific problem, by combining different expressions using genetics-based processes such as mutations and crossovers.

Lastly, hybrid metrics which feed on both analytical- and machine learning-based methods may also be considered, in case the slower evaluation time for the latter tends to hinder too much the speed of mapping algorithms which apply it. Using a sensible strategy, it may be beneficial to apply metrics like ESP in some cases where speed is paramount, and other, possibly slower, but more accurate methods in cases where the metrics model speed is important but not paramount.

Tackling the presented scalability issues, while still achieving high accuracy and reasonable training procedures, could yield a model easily portable to different quantum computing technologies, while empowering mapping algorithms with the ability to make better choices. Achieving such improved mapping procedures can improve the reliability NISQ devices, boosting their capability to solve real world problems in near future.

Bibliography

[1] R. P. Feynman, “Simulating Physics with Computers,” Tech. Rep. 6, 1981. [Online]. Available: https://people.eecs.berkeley.edu/{~}christos/classics/Feynman.pdf

[2] D. Gottesman, “An introduction to quantum error correction and fault-tolerant quantum computation,” 2010, pp. 13–58.

[3] J. Preskill, “Quantum Computing in the NISQ era and beyond,” no. July, pp. 1–20, 2018. [Online]. Available: http://arxiv.org/abs/1801.00862{%}0Ahttp://dx.doi.org/10.22331/q-2018-08-06-79

[4] Glosser.ca, “Bloch Sphere.svg - Wikimedia Commons.” [Online]. Available: https://commons.wikimedia.org/wiki/File:Bloch{_}Sphere.svg [Accessed: 2020-08-03]

[5] F. Marquezino, R. Portugal, and F. Sasse, “Obtaining the Quantum Fourier Transform from the classical FFT with QR decomposition,” *Journal of Computational and Applied Mathematics*, vol. 235, no. 1, pp. 74–81, nov 2010. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0377042710002888

[6] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver, “Superconducting Qubits: Current State of Play,” Tech. Rep., 2019.

[7] X. Zhang, H. O. Li, G. Cao, M. Xiao, G. C. Guo, and G. P. Guo, “Semiconductor quantum computation,” pp. 32–54, 2019.

[8] X. Zhang, H. O. Li, K. Wang, G. Cao, M. Xiao, and G. P. Guo, “Qubits based on semiconductor quantum dots,” p. 20305, 2018. [Online]. Available: http://cpb.iphy.ac.cn

[9] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, “Trapped-Ion Quantum Computing: Progress and Challenges,” apr 2019. [Online]. Available: http://arxiv.org/abs/1904.04178http://dx.doi.org/10.1063/1.5088164

[10] G. J. Milburn and A. G. White, “Quantum computing using optics,” in *Computational Complexity: Theory, Techniques, and Applications*, 2012, vol. 9781461418, pp. 2437–2452.

[11] H. Krovi, “Models of optical quantum computing,” pp. 531–541, mar 2017. [Online]. Available: https://www.degruyter.com/view/journals/nanoph/6/3/article-p531.xml

[12] L. Childress and R. Hanson, “Diamond NV centers for quantum computing and quantum networks,” pp. 134–138, 2013. [Online]. Available: www.mrs.org/bulletin

[13] L. M. K. Vandersypen, C. S. Yannoni, and I. L. Chuang, “Liquid State NMR Quantum Computing,” *ChemInform*, vol. 32, no. 16, pp. no–no, 2010.

[14] V. Lahtinen and J. Pachos, “A Short Introduction to Topological Quantum Computation,” *SciPost Physics*, vol. 3, no. 3, p. 21, 2017.

[15] “Google AI Blog: A Preview of Bristlecone, Google’s New Quantum Processor.” [Online]. Available: https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html [Accessed: 2020-08-02]

[16] R. Versluis, S. Poletto, N. Khammassi, N. Haider, D. J. Michalak, A. Bruno, K. Bertels, and L. Dicarlo, “Scalable quantum circuit and control for a superconducting surface code,” pp. 1–9, 2016.

[17] C. D. Hill, E. Peretz, S. J. Hile, M. G. House, M. Fuechsle, S. Rogge, M. Y. Simmons, and L. C. Hollenberg, “Quantum Computing: A surface code quantum computer in silicon,” *Science Advances*, vol. 1, no. 9, 2015. [Online]. Available: http://advances.sciencemag.org/

[18] “Qubit Quality — Quantum Computing Report.” [Online]. Available: https://quantumcomputingreport.com/scorecards/qubit-quality/ [Accessed: 2019-06-14]

[19] E. Taylor and J. Fortes, “Device Variability Impact on Logic Gate Failure Rates,” *Proceedings of the 16th IEEE Int. Conf. Application-Specific Syst., Archit. & Processors*, 2005.

[20] “ibmq-device-information/backends/tenerife/V1 at master · Qiskit/ibmq-device-information.” [Online]. Available: https://github.com/Qiskit/ibmq-device-information/tree/master/backends/tenerife/V1 [Accessed: 2020-07-31]

[21] D. Greenbaum, “Introduction to Quantum Gate Set Tomography,” 2015. [Online]. Available: http://arxiv.org/abs/1509.02921

[22] E. Nielsen, K. Rudinger, R. Blume-kohout, A. Bestwick, B. Bloom, M. Block, S. Caldwell, M. Curtis, A. Papageorge, A. Polloreno, M. Reagor, N. Rubin, M. Selvanayagam, E. Sete, R. Sinclair, R. Smith, M. Villiers, W. Zeng, and C. Rigetti, “Efficient gate set tomography on a multi-qubit superconducting processor,” *APS*, vol. 2017, p. L46.008, 2017. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2017APS..MARL46008N/abstract

[23] W. K. Wootters and W. H. Zurek, “A single quantum cannot be cloned,” *Nature*, vol. 299, no. 5886, pp. 802–803, 1982. [Online]. Available: https://www.nature.com/articles/299802a0

[24] S. Endo, S. C. Benjamin, and Y. Li, “Practical Quantum Error Mitigation for Near-Future Applications,” *Physical Review X*, vol. 8, no. 3, dec 2018. [Online]. Available: http://arxiv.org/abs/1712.09271http://dx.doi.org/10.1103/PhysRevX.8.031027

[25] A. Kandala, K. Temme, A. D. Córcoles, A. Mezzacapo, J. M. Chow, and J. M. Gambetta, “Error mitigation extends the computational reach of a noisy quantum processor,” *Nature*, vol. 567, no. 7749, pp. 491–495, mar 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1040-7

[26] X. Fu, L. Lao, C. G. Almudever, F. Sebastiano, R. Versluis, E. Charbon, and K. Bertels, “A Heterogeneous Quantum Computer Architecture,” 2016, pp. 1–1.

[27] X. Fu, M. A. Rol, C. C. Bultink, J. Van Someren, N. Khammassi, I. Ashraf, R. F. Vermeulen, J. C. De Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudéver, L. DiCarlo, and K. Bertels, “A Microarchitecture for a Superconducting Quantum Processor,” *IEEE Micro*, vol. 38, no. 3, pp. 40–47, 2018.

[28] N. Khammassi, I. Ashraf, J. v. Someren, R. Nane, A. M. Krol, M. A. Rol, L. Lao, K. Bertels, and C. G. Almudever, “OpenQL : A Portable Quantum Programming Framework for Quantum Accelerators,” may 2020. [Online]. Available: http://arxiv.org/abs/2005.13283

[29] “cirq.Moment — Cirq 0.8.0.dev documentation.” [Online]. Available: https://cirq.readthedocs.io/en/stable/generated/cirq.Moment.html [Accessed: 2020-05-03]

[30] S. G. V. Wee, “Mapping of quantum algorithms on a quantum chip,” 2017.

[31] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, “On the qubit routing problem,” pp. 1–29, 2019. [Online]. Available: http://arxiv.org/abs/1902.08091

[32] S. A. Cook, “The complexity of theorem-proving procedures,” in *Proceedings of the Annual ACM Symposium on Theory of Computing*. New York, New York, USA: Association for Computing Machinery, may 1971, pp. 151–158. [Online]. Available: http://portal.acm.org/citation.cfm?doid=800157.805047

[33] J. Kawahara, T. Saitoh, and R. Yoshinaka, “The time complexity of the token swapping problem and its parallel variants,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10167 LNCS. Springer Verlag, 2017, pp. 448–459.

[34] R. Wille, L. Burgholzer, and A. Zulehner, “Mapping Quantum Circuits to IBM QX Architectures Using the Minimal Number of SWAP and H Operations,” *DAC*, pp. 1–6, 2019.

[35] P. Murali, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, “Formal constraint-based compilation for noisy intermediate-scale quantum systems,” *Microprocessors and Microsystems*, vol. 66, pp. 102–112, 2019.

[36] S. Nishio, Y. Pan, T. Satoh, H. Amano, and R. Van Meter, “Extracting Success from IBM’s 20-Qubit Machines Using Error-Aware Compilation,” 2019. [Online]. Available: http://arxiv.org/abs/1903.10963

[37] A. Zulehner and A. Paler, “Efficient Mapping of Quantum Circuits to the IBM QX Architectures,” pp. 1–4.

[38] A. Zulehner and R. Wille, “Compiling SU (4) Quantum Circuits to IBM QX Architectures,” no. 4.

[39] A. Zulehner, A. Paler, and R. Wille, “An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures,” dec 2017. [Online]. Available: http://arxiv.org/abs/1712.04722

[40] J. X. Lin, E. R. Anschuetz, and A. W. Harrow, “Using Spectral Graph Theory to Map Qubits onto Connectivity-Limited Devices,” 2019. [Online]. Available: http://arxiv.org/abs/1910.11489

[41] G. Li, Y. Ding, and Y. Xie, “Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices,” in *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, 2019, pp. 1001–1014.

[42] D. Venturelli, M. Do, E. Rieffel, and J. Frank, “Compiling quantum circuits to realistic hardware architectures using temporal planners,” *Quantum Science and Technology*, vol. 3, no. 2, pp. 1–31, 2018.

[43] K. E. C. Booth, M. Do, J. C. Beck, E. Rieffel, D. Venturelli, and J. Frank, “Comparing and Integrating Constraint Programming and Temporal Planning for Quantum Circuit Compilation,” 2018. [Online]. Available: http://arxiv.org/abs/1803.06775

[44] D. Venturelli, M. Do, J. Frank, E. Rieffel, K. E. C Booth, T. Nguyen, P. Narayan, and S. Nanda, “Quantum Circuit Compilation: An Emerging Application for Automated Reasoning,” Tech. Rep., 2018. [Online]. Available: https://openreview.net/pdf?id=S1eEBO3nFE

[45] S. Herbert and A. Sengupta, “Using Reinforcement Learning to find Efficient Qubit Routing Policies for Deployment in Near-term Quantum Computers,” vol. 1, pp. 1–13, 2018. [Online]. Available: http://arxiv.org/abs/1812.11619

[46] K. Bertels, C. G. Almudever, L. Lao, I. Ashraf, B. van Wee, N. Khammassi, and J. van Someren, "Mapping of lattice surgery-based quantum circuits on surface code architectures," *Quantum Science and Technology*, vol. 4, no. 1, p. 015005, 2018.

[47] S. S. Tannu and M. K. Qureshi, "A Case for Variability-Aware Policies for NISQ-Era Quantum Computers," pp. 1–12, 2018. [Online]. Available: http://arxiv.org/abs/1805.10224

[48] ——, "Not All Qubits Are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers," in *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, 2019, pp. 987–999.

[49] P. Murali, J. M. Baker, A. J. Abhari, F. T. Chong, and M. Martonosi, "Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers," in *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, 2019, pp. 1015–1029.

[50] A. Van Rynbach, A. Muhammad, A. C. Mehta, J. Hussmann, and J. Kim, "A Quantum Performance Simulator based on fidelity and fault-path counting," p. 13, 2012. [Online]. Available: http://arxiv.org/abs/1212.0845

[51] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[52] "Fly Me To The Moon (Plié) – Sheet music for ballet class — Music for Ballet Class." [Online]. Available: https://www.musicforballetclass.com/product/fly-me-to-the-moon-plie-sheet-music-for-ballet-class/ [Accessed: 2020-05-12]

[53] "Quantum Architectures: qasm2circ." [Online]. Available: https://www.media.mit.edu/quanta/qasm2circ/ [Accessed: 2019-06-14]

[54] "AI could help us deconstruct why some songs just make us feel so good — MIT Technology Review." [Online]. Available: https://www.technologyreview.com/2019/11/01/254/ai-machine-learning-music-feel-good/ [Accessed: 2020-05-12]

[55] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," pp. 1235–1270, 2019.

[56] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A Critical Review of Recurrent Neural Networks for Sequence Learning," 2015. [Online]. Available: http://arxiv.org/abs/1506.00019

[57] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language*

Processing, Proceedings of the Conference, pp. 1724–1734, jun 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[58] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[59] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *Journal of Machine Learning Research*, vol. 3, no. 1, pp. 115–143, 2003. [Online]. Available: www.idsia.ch

[60] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *32nd International Conference on Machine Learning, ICML 2015*, vol. 1. International Machine Learning Society (IMLS), feb 2015, pp. 448–456. [Online]. Available: https://arxiv.org/abs/1502.03167v3

[61] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *35th Conference on Uncertainty in Artificial Intelligence, UAI 2019*, 2019. [Online]. Available: https://github.com/quark0/darts

[62] "Why not Mean Squared Error(MSE) as a loss function for Logistic Regression? — by Rajesh Shreedhar Bhat — Towards Data Science." [Online]. Available: https://towardsdatascience.com/why-not-mse-as-a-loss-function-for-logistic-regression-589816b5e03c [Accessed: 2020-08-03]

[63] "Quantum Inspire." [Online]. Available: https://www.quantum-inspire.com/https://qutech.nl/quantum-inspire/

[64] T. Last, N. Samkharadze, P. Eendebak, R. Versluis, X. Xue, A. Sammak, D. Brousse, K. Loh, H. Polinder, G. Scappucci, M. Veldhorst, L. Vandersypen, K. Maturova, J. Veltin, and G. Alberts, "Quantum Inspire: QuTech's platform for co-development and collaboration in quantum computing," in *Novel Patterning Technologies for Semiconductors, MEMS/NEMS and MOEMS 2020*, E. M. Panning and M. I. Sanchez, Eds., vol. 11324, no. 23. SPIE, mar 2020, p. 17. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/11324/2551853/Quantum-Inspire--QuTechs-platform-for-co-development-and-collaboration/10.1117/12.2551853.full

[65] "Qiskit." [Online]. Available: https://qiskit.org/ [Accessed: 2020-08-03]

[66] N. Khammassi, I. Ashraf, X. Fu, C. G. Almudever, and K. Bertels, "QX: A high-performance quantum computer simulation platform," in *Proceedings of the 2017 Design, Automation and Test in Europe*. Institute of Electrical and Electronics Engineers Inc., may 2017, pp. 464–469.

[67] R. LaRose, “Overview and Comparison of Gate Level Quantum Software Platforms,” *Quantum*, vol. 3, p. 130, 2019. [Online]. Available: https://www.research.ibm.com/ibm-q/network/.

[68] T. E. O’Brien, B. Tarasinski, and L. DiCarlo, “Density-matrix simulation of small surface codes under current and projected experimental noise,” pp. 1–9, 2017. [Online]. Available: http://arxiv.org/abs/1703.04136

[69] T. O’Brien, “GitHub - quantumsim/qsoverlay: An overlay for the quantumsim package developed by Brian Tarasinski (https://github.com/brianzi/quantumsim).” [Online]. Available: https://github.com/quantumsim/qsoverlay [Accessed: 2020-10-09]

[70] C. C. Lin, A. Chakrabarti, and N. K. Jha, “QLib: Quantum module library,” *ACM Journal on Emerging Technologies in Computing Systems*, vol. 11, no. 1, pp. 1–20, oct 2014. [Online]. Available: https://dl.acm.org/doi/10.1145/2629430

[71] “RevLib - An Online resource for Reversible Benchmarks.” [Online]. Available: http://www.revlib.org/ [Accessed: 2020-08-03]

[72] “TensorFlow.” [Online]. Available: https://www.tensorflow.org/ [Accessed: 2020-08-03]

[73] T. Y. Liu, “Learning to rank for Information Retrieval,” *Foundations and Trends in Information Retrieval*, vol. 3, no. 3, pp. 225–231, 2009. [Online]. Available: http://dx.doi.org/10.1561/1500000016

[74] M. G. Kendall, “A New Measure of Rank Correlation,” *Biometrika*, vol. 30, no. 1/2, p. 81, jun 1938.

[75] Y. Dodge, “Kendall Rank Correlation Coefficient,” in *The Concise Encyclopedia of Statistics*. Springer New York, feb 2008, pp. 278–281. [Online]. Available: https://link.springer.com/referenceworkentry/10.1007/978-0-387-32833-1{_}211

[76] ——, “Spearman Rank Correlation Coefficient,” in *The Concise Encyclopedia of Statistics*. Springer New York, feb 2008, pp. 502–505. [Online]. Available: https://link.springer.com/referenceworkentry/10.1007/978-0-387-32833-1{_}379

[77] R. Sun, “Optimization for deep learning: theory and algorithms,” 2019. [Online]. Available: http://arxiv.org/abs/1912.08957

[78] S. Doshi, “Various Optimization Algorithms For Training Neural Network — Medium.” [Online]. Available: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6https://medium.com/@sdoshi579/optimizers-for-training-neural-network-59450d71caf6 [Accessed: 2020-08-19]

[79] R. Wierenga, “An Empirical Comparison of Optimizers for Machine Learning Models — Heartbeat.” [Online]. Available: https://heartbeat.fritz.ai/an-empirical-comparison-of-optimizers-for-machine-learning-models-b86f29957050 [Accessed: 2020-08-19]

[80] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[81] J. Kiefer and J. Wolfowitz, “Stochastic Estimation of the Maximum of a Regression Function,” *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, sep 1952. [Online]. Available: https://projecteuclid.org/euclid.aoms/1177729392

[82] S. Ruder, “An overview of gradient descent optimization algorithms,” 2016. [Online]. Available: http://caffe.berkeleyvision.org/tutorial/solver.htmlhttp://arxiv.org/abs/1609.04747

[83] L. N. Smith, “Cyclical learning rates for training neural networks,” in *Proceedings - 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*. Institute of Electrical and Electronics Engineers Inc., jun 2017, pp. 464–472. [Online]. Available: http://arxiv.org/abs/1506.01186

[84] “Estimating an Optimal Learning Rate For a Deep Neural Network.” [Online]. Available: https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0 [Accessed: 2020-05-18]

[85] G. Graziano, “Quantum information leakage,” *Nature Reviews Chemistry*, vol. 4, no. 4, p. 170, apr 2020. [Online]. Available: https://doi.org/10.1038/s41570-020-0178-z

[86] M. Sarovar, T. Proctor, K. Rudinger, K. Young, E. Nielsen, and R. Blume-Kohout, “Detecting crosstalk errors in quantum information processors,” *Quantum*, vol. 4, 2020.

[87] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A Comprehensive Survey on Graph Neural Networks,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2020.

[88] A. Ajit, K. Acharya, and A. Samanta, “A Review of Convolutional Neural Networks,” in *International Conference on Emerging Trends in Information Technology and Engineering, ic-ETITE 2020*. Institute of Electrical and Electronics Engineers Inc., feb 2020.

[89] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, vol. 2017-Decem, 2017, pp. 5999–6009.

[90] N. Kitaev, Ł. Kaiser, and A. Levskaya, “Reformer: The Efficient Transformer,” jan 2020. [Online]. Available: http://arxiv.org/abs/2001.04451

[91] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, “Validating quantum computers using randomized model circuits,” *Physical Review A*, vol. 100, no. 3, Sep 2019. [Online]. Available: http://dx.doi.org/10.1103/PhysRevA.100.032328

[92] B. Jónsson, B. Bauer, and G. Carleo, “Neural-network states for the classical simulation of quantum computing,” pp. 1–8, 2018. [Online]. Available: http://arxiv.org/abs/1808.05232

[93] W.-Y. Loh, “Classification and Regression Trees,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, pp. 14–23, 2011.

[94] M. T. Ahvanooey, Q. Li, M. Wu, and S. Wang, “A survey of genetic programming and its applications,” *KSII Transactions on Internet and Information Systems*, vol. 13, no. 4, pp. 1765–1794, apr 2019.

[95] T. J. Sejnowski, “The unreasonable effectiveness of deep learning in artificial intelligence,” *Proceedings of the National Academy of Sciences*, p. 201907373, jan 2020. [Online]. Available: www.pnas.org/cgi/doi/10.1073/pnas.1907373117

A

Quantum Computer Implementations

Class	Structure where qubits are encoded	Qubit Type
Superconducting	Superconducting Loops	Charge
		Flux
		Phase
		Transmon
		Fluxon
Semiconductor	Quantum dots (in silicon)	Charge
		Spin (Loss-DiVincenzo)
		Singlet-triplet
		Exchange
		Hybrid
	Phosporus donor atoms, in silicon (Kane QC)	Donor's electron/nuclear spins; electric dipole
	Nitrogen-vacancy centers (in diamond)	Electron/nuclear spins
Ultracold atoms/ions	Ions (usually Group-II ions) in Ion Traps (either Paul or Penning traps)	Zeeman
		Hyperfine
		Optical
	Rydberg atoms in an optical lattice	Hyperfine, Zeeman, etc...
NMR	Ensemble of molecules	Nuclear Spins
Quantum Optics	Linear optical elements	Photon polarization
		Photon frequency

Table A.1: Breakdown of the most popular quantum computer concepts.

B

Routing algorithm frameworks

There are two main frameworks based on which the routing problem is usually approached, that serve as base for routing algorithms: the SWAP-based framework and the Mapper-Permuter framework.

B.1 SWAP framework

The SWAP-based framework envisions the routing problem as a matter of, given a circuit and an initial mapping, adding SWAP gates in a sensible way, such that the resulting circuit respects the device's connectivity constraints. This is not to be confused with the SWAP-based method of routing (Sec. 2.4.2). Typically, SWAP-framework-based algorithms follow this outline:

1. The input circuit is divided in slices (windows) according to an arbitrary criteria (ex. cycles).
2. Determine the best sequence of SWAPS the operands of upcoming two-qubit gates must undergo such that, by the end of each round, the connectivity constrains are respected.

This procedure is open-ended in the sense that the final mapping (at the end of each round) is not predefined, rather, any mapping that respects the connectivity constraints is valid.

B.2 Mapper-Permuter framework

The Mapper-Permuter framework takes the SWAP framework as basis for a more complex framework which, contrary to the SWAP framework, is not open-ended: an optimal final mapping is computed for each round, and the SWAP scheme has to change the mapping in a way that, not only the connectivity constraints are respected, but the final mapping is a predefined one. It follows the steps:

1. The input circuit is divided in slices according to an arbitrary criteria (ex. cycles).
2. Determine the best mapping for each slice, such that all the gates inside are connectivity-compliant.
3. For each slice S_i, insert before it a new slice S_{i*} and populate it with a set of a permutations such that the previous slice's (S_{i-1}) mapping is transformed to the one generated in the previous step. This set of permutations may be, for example, calculated by a Token Reconfiguration-specific tool. The permutations are implemented via SWAP gates.

This framework essentially encapsulates the SWAP-framework, but ads a layer of complexity, by computing an optimal final mapping for each round. A good mapper-permuter algorithm will aim to make temporally-adjacent mappings similar, in order to minimize the amount of permutations needed at every step.