Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

An Approach for Benchmarking Quantum Computers to Determine the Executability of Quantum Circuits

Jan Weber

Course of Study: Softwaretechnik

Examiner: Prof. Dr. Dr. h.c. Frank Leymann

Supervisor: Marie Salm, M.Sc.,
Benjamin Weder, M.Sc.

Commenced: November 25, 2020

Completed: May 25, 2021

Kurzfassung

Quantencomputer sind derzeit noch stark durch ihre hohe Fehlerraten und ihre niedrigen Anzahl an Qubits beschränkt. Dies verursacht Fehler bei der Ausführung von Quantenalgorithmen, weswegen man nach aktuellem Stand nicht erwarten kann, dass ein beliebiger Schaltkreis fehlerfrei ausgeführt wird. Die Größe eines Schaltkreises beeinflusst die Genauigkeit des Ergebnisses bedeutsam, wobei größere Schaltkreise wesentlich anfälliger für Fehler sind. Die Größe eines Schaltkreises ist definiert durch wd, wobei w die Breite und d die Tiefe angibt. Es existieren Metriken, die es erlauben die Leistungsfähigkeit von Quantencomputern zu bewerten und Vorhersagen zu treffen, ob ein bestimmter Schaltkreis ausführbar ist oder nicht. Für diese Arbeit wurden randomisierte Schaltkreise auf gatter-basierten Quantencomputern ausgeführt und die Ergebnisse mit denen des Quantensimulators verglichen. Um zu bewerten, ob ein Benchmark erfolgreich war oder nicht, wurden vier Metriken in Betracht gezogen. Von diesen Metriken erweist sich die Histogram Intersection Metrik als die beste Methode um die Qualität des Ergebnisses zu bewerten. Mit Hilfe dieser Metrik ist es nun möglich, die Quantencomputer mit randomisierten Schaltkreisen von unterschiedlicher Größe zu benchmarken und die Ergebnisse zu evaluieren, um eine maximal mögliche Schaltkreisgröße zu bestimmen. Für IBMs Quantencomputer *imbq_athens* ergibt sich bei Schaltkreisen mit geringer Breite von 1 bis 3 eine Größe von $wd \leq 20$, innerhalb der noch akzeptable Ergebnisse erwartet werden können. Schaltkreise, die 4 oder 5 Qubits nutzen, erlauben sogar Schaltkreise mit einer Größe $wd \leq 40$. Mit dem bereitgestellten Framework können Benchmarks ausgeführt werden, mit denen die Metrik $wd < k\frac{1}{\epsilon_{\text{eff}}}$ ermittelt werden kann. Diese Metrik soll bei der automatisierten Auswahl eines passenden Quantencomputers für einen gegebenen Schaltkreis genutzt werden.

Abstract

Quantum computers in this day and age are characterized by high error rates and their limited amount of qubits. This introduces errors to the execution of quantum circuits. Consequently, quantum computers currently cannot be expected to run arbitrary circuits successfully. In this context, the size of a circuit heavily influences the outcome of the execution, as large circuits are prone to errors. The size of a circuit is defined by wd where w is its width and d its depth. Metrics can be used to judge the computational power of quantum computers and allow predictions on whether a circuit is expected to run successfully or not. In this thesis, gate-based quantum computers were benchmarked by executing randomized circuits and comparing the results to the quantum simulator's result. Four different metrics were considered to evaluate whether the quantum computer's result is too erroneous to consider the benchmark successful or not. After comparing the metrics and discussing possibilities as to how they can be used to evaluate benchmarks, it was decided that the histogram intersection is the most appropriate to use. Using this metric, it is possible to benchmark quantum computers with randomized circuits of different sizes, evaluate the results and use that data to find upper limits on the circuit size. The data in this thesis suggests that, for IBM's quantum computer *imbq_athens*, circuits of size $wd \leq 20$ are expected to return acceptable results while circuits of width equal to 4 or 5 deliver acceptable results for even larger circuits (up to $wd = 40$). The framework provided in this thesis is the foundation to determine the metric $wd < k\frac{1}{\epsilon_{\text{eff}}}$ which will be used in the automated selection of an appropriate quantum computer for a given quantum circuit.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

While the error rates of today's quantum computers are still high and introduce errors to the computation, the strongly limited number of qubits also limits the maximum size of a circuit [Pre18]. Because of these limitations, not every quantum circuit is guaranteed to run successfully on a quantum computer. This raises interest in metrics that predict the severity of the error in the result before executing a given quantum circuit on a specific quantum computer [SBLW20]. Metrics like the Total Quantum Factor [SZR16] or Quantum Volume [MBB+18] use the size of the circuit, defined by its width and depth, and the error rate of the quantum computer, consisting of different hardware properties, to give an estimate on the maximum circuit size a quantum computer can successfully handle [SZR16] [MBB+18]. Another metric, the so-called rule of of thumb, is used to predict whether a circuit of width w and depth d can be executed on a quantum computer with error rate ϵ_{eff} [LB20] [Pre18]:

$$wd \ll \frac{1}{\epsilon_{\text{eff}}}$$

However, this is a very rough estimate and it is desirable to sharpen it for a more precise prediction [SBLW20]. The rule of thumb suggests that there is a point $\frac{1}{\epsilon_{\text{eff}}} - \lambda$ that marks the switch from successful to failed execution, where λ is so far unknown [SBLW20]. This means that the interval $[0, \frac{1}{\epsilon_{\text{eff}}} - \lambda]$ indicates the range of accepted values. It is possible to replace $\frac{1}{\epsilon_{\text{eff}}} - \lambda$ by $k\frac{1}{\epsilon_{\text{eff}}}$ which allows to state a more precise metric [SBLW20]:

$$wd < k\frac{1}{\epsilon_{\text{eff}}}$$

Having a precise prediction is particularly important for automating the selection of a quantum computer for a specific quantum circuit which is proposed in [SBB+20]. Otherwise the confidence in the prediction is low and errors might be introduced again. Computing the maximum size of a circuit is the first step to sharpen the metric as mentioned above. To compute that limit, quantum computers need to be benchmarked and the data needs to be evaluated to determine the values of k, ϵ_{eff} and λ [SBLW20].

In this thesis, randomized circuits of different sizes are used to benchmark quantum computers from IBM, specifically *ibmq_athens* [1], and determine the maximum possible size of a circuit that a quantum computer can successfully execute. For this purpose, it is necessary to use metrics that allow to evaluate the quantum computer's result. The correct result was obtained with IBM's quantum simulator *ibmq_qasm_simulator* [2] and the histograms of the results of the quantum computer and simulator were compared. Different metrics were used with the intersection metric being the most meaningful.

[1] https://quantum-computing.ibm.com/services?system=ibmq_athens&systems=all

[2] https://quantum-computing.ibm.com/lab/docs/iql/manage/simulator/#ibmq-qasm-simulator

The following steps were taken in this thesis: First, a framework had to be implemented that allows to easily execute randomized quantum circuits on quantum computers and compare their output to the correct results. Using this framework, benchmarks were run and the data saved in a database. Finally, the data from the database was analysed and evaluated.

The thesis is structured as follows: In the beginning, general background on quantum computers and necessary fundamentals are covered in Chapter 2. Related work on benchmarking quantum computers is covered in Chapter 3. The research design is explained in Chapter 4. The implementation of the benchmarks is described in Chapter 5 and Chapter 6. The data is analysed in Chapter 7. The last chapter summarizes the work and gives an outlook.

2 Background

This chapter covers the fundamentals of this thesis. We start with a general overview on quantum computing that is necessary to be able to follow the approach that was taken. With that knowledge it is possible to take a closer look at the so-called *Noisy Intermediate-Scale Quantum (NISQ)* devices which are the most common type of quantum devices at this time. Finally, information on IBM's quantum devices and the toolkit that was used and extended is provided.

2.1 General

Classical computers store information in bits [Ker19]. Each bit can either be in state 1 or 0 and by concatenating the state of several bits with each other and by manipulating their states, it is possible to perform calculations and create complex applications, e.g. to edit images [Ker19]. The concept of a bit can be realized in different ways on actual hardware. Looking at processors and their cache, very small electric charges are used to realize the two states 0 and 1 [Ker19] [MN19].

Quantum computers use so-called *quantum bits*, or simply *qubits* [RP11]. In contrast to classical computers, the state is not a simple number but a vector. The two-dimensional vector space is called *state space* [NC10]. To understand what it means that a state is represented by a two-dimensional vector, take a look at the *computational basis* of the state space [MN19]:

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

The *ket* notation is typical for quantum states. That being said, these two vectors behave like the 0 and 1 bits in classical computing. The state space for quantum computing is complex and it is possible to create *superpositions* of the basis states $|0\rangle$ and $|1\rangle$. A superposition is simply a linear combination [MN19]:

$$\alpha\,|0\rangle + \beta\,|1\rangle = \alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}\alpha\\\beta\end{bmatrix}$$

The *amplitude* of a state is the coefficient of a state in a superposition [RP11]. Looking at the equation above, the amplitude for $|0\rangle$ is α. With this, it is possible to define the *normalization constraint* of a quantum state as not every vector is accepted as a state vector. *It is necessary that the sum of the squared amplitudes equals 1* [MN19]. Keeping in mind that the amplitudes could be complex numbers this constraint is denoted by the following equation: $|\alpha|^2 + |\beta|^2 = 1$ [NC10]. It is not possible to simply measure a qubit to read the amplitudes [MN19]. Instead, when you measure a qubit in state $\alpha\,|0\rangle + \beta\,|1\rangle$, you expect to get 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$. Note that these are classical bits again [RP11]. Measuring a qubit also puts it back into one of the two computational basis states $|0\rangle$ or $|1\rangle$ depending on the classical bit the measurement returns and information about α and β are lost [RP11].

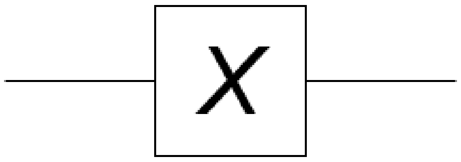


Figure 2.1: Quantum Circuit with NOT Gate

To manipulate the state of qubits, *quantum logic gates* are used [RP11]. Quantum gates can be represented by matrices, e.g. the X gate which is the analogue of the classical NOT gate [MN19]:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

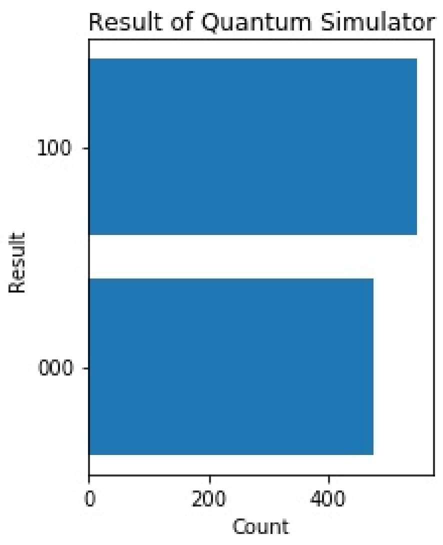
Quantum computers that use gates for their operations, are called *gate-based* [NC10] quantum computers. A gate can be represented graphically, usually as a box with a letter that specifies the type of gate, e.g. X [RP11]. Several quantum gates as a sequence form a *quantum circuit* [NC10], similar to circuits in classical computing [NC10]. For each qubit, a horizontal line, the *quantum wire* is shown on which the gate operations on this specific qubit over time are indicated [MN19]. Figure 2.1 shows a quantum circuit with only one qubit and the NOT gate.

At this point it is interesting to note that it is still very difficult to control the qubits [NC10]. This means that a quantum circuit consisting of only one quantum wire, which seems to be an easy circuit, is actually difficult to realize. It would require a steady control over the qubit for the whole time but the state is very fragile and a small interference can already change the state [MN19].

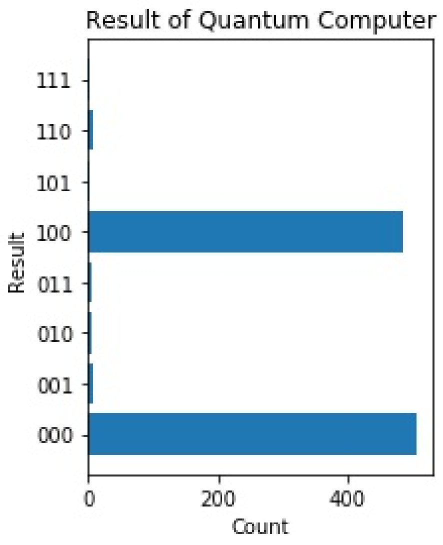
So far, the focus was on single-qubit operations but multi-qubit systems are built analogously [MN19]. For example, a two-qubit system has four computational basis states: $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ [NC10]. Just like in the single-qubit case, it is possible to put the two qubits in superposition and the squared coefficients have to add up to 1 again [RP11]. With that in mind, it is possible to take a general look at quantum computing: First, the qubits are initialized in a computational basis state. Afterwards, the circuit is ran by executing the given sequence of gates. By measuring the qubits like mentioned before, it is possible to obtain the result [MN19].

Since quantum computers deliver erroneous results and it is not possible to measure the exact amplitudes of a qubit, it is necessary to execute circuits many times which provides a probability distribution of the different results [MNW+17]. For IBM, the number of executions is called *shots* and each execution might return a different or the same result. By executing a circuit a specific amount of times, it is possible to calculate probabilities for each result. As for IBM, the *counts* for all bit strings that represent the state of the quantum computer's qubits after the execution are the actual result of the execution [MNW+17]. The results can be represented as histograms, where each bit string in the result has a bar representing its count. Figure 2.2 shows the results of the execution of Shor's factorization algorithm [Bea03] for a fixed input of 15 [1] visualized as histograms. The circuit was executed 1024 times on both backends and depending on how often each bit string was returned, its bar is smaller or larger in the histogram. While Figure 2.2a shows the result of IBM's quantum simulator *ibmq_qasm_simulator*, Figure 2.2b is the visualization for the result

[1]Implementation: `https://github.com/UST-QuAntiL/nisq-analyzer-content/blob/master/example-implementations/Shor/shor-fix-15-qiskit.py`



(a) Histogram of Execution on *ibmq_qasm_simulator*



(b) Histogram of Execution on *ibmq_athens*

Figure 2.2: Histograms for Shor Algorithm with Input 15

of the quantum computer *ibmq_athens*. Having the two histograms, it is possible to compare the results. We can assume that the simulator's result is correct [IBM21] while the quantum computer is, as already stated, prone to errors. Looking at the two histograms, one obvious difference is that the quantum computer returned 000 more often than 100, while it was the other way around for the simulator. Furthermore, the quantum computer returns additional bit strings like 111 that do not occur at all in the simulator's result. The transpiled circuit on the quantum computer has a width equal to 5 and depth equal to 7 but it is optimized for the purpose to determine the factorization of 15 and because of that it is expected to return a result close to the correct one. In this case, the results are very similar. Bit strings that should not be returned, appeared only in a small fraction of the executions, while 000 and 100 still make the majority of the result. This is also the motivation for benchmarking quantum computers as proposed in this thesis. It is an approach to test the performance of current quantum computers not by theoretical models but by executing circuits which will also take real world interferences and limitations into account.

Having provided the basics of quantum computing one might ask oneself whether there is any advantage in using quantum computers over classical computers, whether there is a real life scenario where they are superior to classical computers. Simulating quantum systems, e.g. molecules, is a difficult task for classical computers as the number of amplitudes that need to be stored rise exponentially [MN19]. For a quantum computer however, the amount of extra qubits needed is a small number hence they are much better suited for the simulation of quantum systems. The use of that in applications might not be immediately obvious but it can be of significant meaning in chemistry for pharmaceutical purposes or simulations in quantum physics [MN19].

2.2 Noisy Intermediate-Scale Quantum Devices

Most of today's quantum computers are so-called *Noisy Intermediate-Scale Quantum (NISQ)* devices [Pre18] [LB20]. Noisy means that the operations on and the states of the qubits are erroneous. The decay of a qubit's state is known as *decoherence* [RP11]. At this point in time, it is not possible to ensure that the manipulation of qubits by the gates are flawless, hence there might be differences from the ideal scenario that influence the final result as well. This is known as *gate infidelity* [RP11] [NC10]. Intermediate-Scale indicates that the number of qubits is still very limited [LB20].

These two factors play an important role to determine whether a circuit can be successfully executed on a quantum computer [LB20]. Noise is likely to increase with the depth of a circuit. The depth of a circuit is given by the number of gates executed sequentially [NC10]. Since the circuit would take longer to execute, the decay of the qubit's state has more time to have an effect on the result. The operations on the qubit introduce further errors and the more operations are executed, the larger is their effect [NC10]. Furthermore, the width of the circuit is limited by the number of qubits, as the width indicates the number of required qubits [LB20].

In [Pre18], Preskill elaborates on the impact of NISQ devices on quantum computing in general. While NISQ will not immediately be able to prove the quantum speedup, Preskill sees those devices as an important milestone. Since it is now possible to experiment and work with real quantum computers, it is likely that better and more quantum algorithms will be developed. With more time spent on the process of controlling the state of qubits, methods to reduce the noise in the execution can be developed. Similarly, the accuracy of a quantum gate's operations is likely to be improved and have a positive effect on the maximum circuit size [Pre18]. Hence, it will be interesting to see the developments of NISQ devices in the coming years. If the number of qubits can be increased and the noise decreased, NISQ devices can be a very important factor in quantum computing [Pre18]. In fact, researchers at Google managed to show quantum speedup in 2019 and published an article [2]. A task that would take a classical computer thousands of years was solved within a few minutes.

At this point in time, many different vendors are starting to provide quantum computers independently. The vendors grant access to their backends in the cloud most of the time [SBB+20]. Each vendor develops its own *Software Development Kit (SDK)* that is used to implement circuits and run them on a quantum computer [SBB+20]. Different quantum computers have a different amount of qubits and are also distinguished by their implemented gates, known as the *gate set* [LB20]. These differences implicate that not every SDK from any vendor is compatible with every quantum computer [SBB+20]. [SBB+20] notes that while there are SDKs available that can transpile circuits for quantum computers of different vendors, there is still a significant amount of effort necessary to use those.

[2]https://www.nature.com/articles/s41586-019-1666-5

2.3 IBM Quantum

For this thesis, gate-based quantum computers from IBM were used. IBM is a provider for quantum computers since 2016 [MNW+17]. They provide cloud access to their quantum computers to the public via IBM Quantum [3] and have their own SDK, called Qiskit [4] [IBM21] [AAA+19]. IBM Quantum grants access to a variety of quantum devices, both quantum simulators and quantum computers. The quantum computers vary in the number of qubits, their *topology*, which is the arrangement of the qubits, as well as processor type and error rates [IBM21]. Qiskit allows users to implement quantum circuits with Python, and run them on real backends, either a quantum simulator or a quantum computer [AAA+19]. To execute a circuit on a backend, a job is created and placed in a queue. As soon as the jobs ahead are done, the job is executed on the quantum device and the result will be returned to the user. The circuits need to be transpiled for a specific backend. Since the topology is different depending on the chosen backend, circuits have to be individually mapped on the hardware of each backend [AAA+19]. This individual mapping ensures that the circuit is running as fast and with as little flaws as possible. The translation is known as *transpilation* [SBB+20]. IBM uses OpenQASM [5] as assembly language [CBSG17].

In this thesis, the focus was on IBM's quantum computer *imbq_athens* [6]. An important reason for that was simply the availability, as the queues were rather empty compared to other quantum computers. It is a 5 qubit quantum computer with a quantum volume of 32 and one of IBM's Falcon processors [IBM21].

As mentioned before, IBM's quantum simulator *ibmq_qasm_simulator* [7] can be used to obtain the correct result of a circuit. The simulator can also model noise to create erroneous results like real quantum computers. The maximum amount qubits that can be simulated is 32 for this specific simulator and it has a timeout mechanic is implemented that limits the maximum amount of time a job can take to run [IBM21]. This means that the simulator cannot simulate arbitrarily sized circuits but is limited by the maximum amount of qubits and the maximum execution time.

2.4 Qiskit-Service

The *qiskit-service* [8] is a toolkit that uses Qiskit and provides an API to transpile and execute quantum circuits on IBM quantum services. The transpilation request of this service takes a Qiskit or OpenQASM implementation provided either directly as data or as an URL and returns the transpiled OpenQASM string as well as the width and depth of the circuit. To run the execution request, the implementation can be provided again as Qiskit code (which would run the tranpilation request first) or directly as an OpenQASM string. The execution request naturally returns the results of the execution, i.e. the counts for the resulting qubit states.

[3] https://quantum-computing.ibm.com/

[4] https://qiskit.org

[5] https://github.com/Qiskit/openqasm

[6] https://quantum-computing.ibm.com/services?system=ibmq_athens&systems=all

[7] https://quantum-computing.ibm.com/lab/docs/iql/manage/simulator/#ibmq-qasm-simulator

[8] https://github.com/UST-QuAntiL/qiskit-service

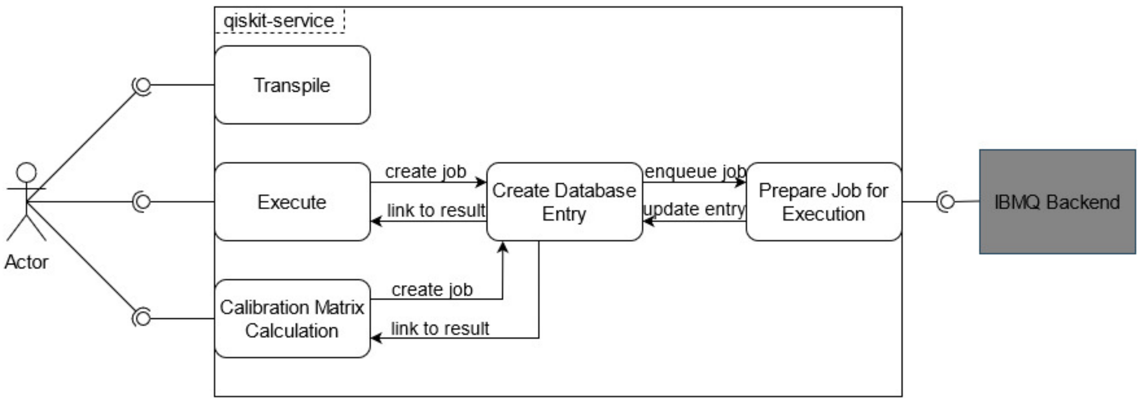


Figure 2.3: Workflow of the qiskit-service

Figure 2.3 shows the current workflow of the qiskit-service. The actor has access to the toolkit via its API which allows to get access to the aforementioned transpilation and execution functionalities. Another API endpoint allows users to retrieve the calibration matrix of a specific quantum computer. While the transpilation request is handled directly within the API, both the execution and calibration matrix request create entries in the database. For these two requests, a job that can be sent to the IBMQ backend needs to be created and put into the queue for that backend. Once this job is executed, the result is returned to the qiskit-service and the actor can retrieve it by following the link provided in the response to the request.

3 Related Work

There is a lot of work dealing with benchmarking quantum computers. Main focus of those benchmarks is to determine the computing power of quantum computers. In this chapter, an overview on other work on benchmarking quantum computers is provided and similarities and differences are presented.

3.1 Benchmarking

[SBLW20] takes a look at three metrics that can be used to judge the capabilities of a quantum computer. Two of those metrics, the *Total Quantum Factor (TQF)* and the *Quantum Volume (QV)*, focus on the performance of a quantum computer itself.

Increasing the circuit size is likely to cause more errors in the result. The TQF defines an upper bound for the circuit size to prevent very erroneous results [SZR16]:

$$TQF := \frac{T_1}{t_g} \cdot n_q$$

The average decoherence time of the qubits is given by T_1, while t_g is the maximum time the available gates need to operate and n_q indicates the number of qubit [SBLW20].

The QV is defined by the following formula [MBB+18]:

$$V_Q = \max_{n' \leq n} \min(n', \frac{1}{n' \epsilon_{\text{eff}}(n')})^2$$

In this equation, n' indicates the circuit's width and n is the number of qubits of the quantum computer. ϵ_{eff} is the effective error rate of the quantum computer and represents average error rate of a two-qubit gate [SBLW20].

The QV is also subject of [CBS+19]. By executing randomized, squared circuits on quantum computers, the performance of current quantum computers is being evaluated. Effectively, the QV, like the TQF, gives an estimate on the maximum circuit size a quantum computer can execute. Note that the QV assumes that the circuits are of equal width and depth. It does however, take *hardware parameters* [CBS+19] like the effective error rate and *design parameters* [CBS+19] like the gate set into account which allows a more precise prediction. It can be used for any quantum service that can execute quantum circuits as it is independent from the architecture [CBS+19] [SBLW20].

Main focus of [SBLW20] is a different metric, though. It is a metric that allows to predict whether an arbitrary given circuit can be successfully run on a quantum computer or not. [Pre18] and [LB20] discuss this metric as a *rule of thumb* [SBLW20]:

$$wd \ll \frac{1}{\epsilon_{\text{eff}}}$$

w and d define the size of the circuit as *width* and *depth*. Like for the QV, ϵ_{eff} is the effective error rate of the quantum computer. This error rate is influenced by various factors [SBLW20]. Salm et al. suggest sharpening this metric by benchmarking quantum computers. The goal is to specify a *refined quantum metric* by extracting properties of the quantum computer at the time of execution and executing different kinds of circuit, e.g. randomized [SBLW20]:

$$wd < k\frac{1}{\epsilon_{\text{eff}}}$$

This is where this thesis comes into play, providing a framework to execute randomized circuits to perform benchmarks.

[BY20] provides a framework for so-called *volumetric benchmarks*. These are benchmarks that generalize IBM's quantum volume benchmarks as they do not only include circuits of same width and depth. Beyond these square circuits, rectangular circuits are accepted which is often a closer representation of real world applications. The authors suggest using randomized, periodic and application circuits to benchmark quantum computers. The different classes have different properties that have to be kept in mind, e.g. periodic circuits can amplify coherent errors over time [BY20]. There are also success criteria for a benchmark being discussed. Expecting a perfect result is not realistic at this point in time and there should be tolerance for some error. If the outcome is definite, success could be defined by "the probability of the correct outcome of circuit C is greater than a specific threshold (e.g. 2/3) with high (e.g. 95%) statistical confidence" [BY20]. A success criterion for all circuits, not just definite, would be checking whether the distribution of the results of a benchmark is within the optimal distribution for the executed circuit [BY20]. However, their work is focused on providing the framework to create and evaluate benchmarks. No benchmark was actually run but they introduced an approach to evaluate data from executions by model data [BY20]. While randomized benchmarks are used in this thesis as well, we need real data to be able to determine the maximum circuit size for a quantum computer.

Benchmarking quantum computers is also discussed in [MNW+17]. It should be noted that this work was published when public access to quantum computers was relatively new. Consequently, researchers were just starting to validate quantum computers independently from the providers of quantum computers. They are also executing a variety of circuits on quantum computers and compared the output from the quantum computer with the expected output deduced from quantum theory [MNW+17]. They conclude that quantum computers at that time did not perform as well as one might expect from a computer [MNW+17]. Tracking down the exact reasons for the errors was not possible but it is likely that the calibration of the machine was a deciding factor. This means that results potentially would have been less erroneous if the circuits were executed on a freshly calibrated machine [MNW+17]. However, the accuracy of quantum computers has improved over the years and they can deliver good results for circuits up to a certain size [LB20] [Pre18].

We are now taking a look again at the amount of different SDKs that was introduced in Chapter 2. The main problem is that the great variety of vendors that each independently develop their own hardware and software causes difficulties in the access of quantum computers as it varies greatly between vendors. In [SBB+20], an approach to automate the selection of a specific quantum computer to run a quantum algorithm, called the NISQ Analyzer, is provided. The steps that need to be taken and expected challenges are part of [SBB+20]. An important part here are metrics that can estimate whether a circuit can be executed successfully or not. This is where the rule of thumb metric and its refined version are important, as they can provide an estimate on exactly that [SBB+20] [SBLW20]. The selection of a specific quantum computer is not trivial and this thesis is an important step to make it possible.

Similarly to that, a retargetable compiler for NISQ devices has been developed [SDC+20]. There is a user interface to create circuits and execute them on quantum computers. It is retargetable, which means that the `t|ket⟩` compiler is able to prepare the input not just for quantum computers of one specific vendor, but different vendors without having to manually re-implement the circuits. They have also performed benchmarks to show that the compiler does improve the circuits to run more efficiently and it is not having a bad impact on the performance [SDC+20]. In this thesis, benchmarks are not used to compare two different compiler options but to judge the quality of the result of a quantum computer.

3.2 Problem Statement

We have seen that a lot of work on benchmarking quantum computers is available. In this thesis we want to find a way to determine the maximum size of a circuit a specific quantum computer can execute successfully. There are several difficulties to implement this. First, because randomized circuits are used, it is necessary to retrieve the correct result by executing the given circuit on a quantum simulator. The correct result is necessary to assess the quality of the quantum computer's result. This difficulty is easy to overcome as IBM also offers several simulator backends [IBM21]. Apart from that, it is also difficult to decide whether a given result is close enough to the correct one or not. It requires further consideration at what point a result deviates too much from the correct result to mark it as successful which turned out to be a major difficulty in this thesis. The idea is to create a framework to execute benchmarks easily. After running many benchmarks with circuits of different sizes, the data needs to be analysed. Four metrics, which are introduced in the next chapter, are used to quantify the quality of the results. By manually comparing the values of these metrics for many benchmarks, a success criterion was deduced which was then used to make an approach to determine the maximum circuit size for a specific quantum computer.

4 Methodology

We will now take a closer look at the approach that was taken to design the framework and the metrics that were used to quantify the quality of the quantum computer's result. The idea is to present a method that allows to determine the maximum circuit size a specific quantum computer can execute successfully. This boundary value will be useful in determining the value $\frac{1}{\epsilon_{\text{eff}}}$ in the refined rule of thumb metric.

4.1 General

In the last section, the rule of thumb metric was introduced [SBLW20]:

$$wd \ll \frac{1}{\epsilon_{\text{eff}}}$$

It is a very rough estimate on whether a circuit of width w and depth d can be executed on a quantum computer with error rate ϵ_{eff}. In fact, this equation implies that there is a point $\frac{1}{\epsilon_{\text{eff}}} - \lambda$ in the interval $[1, \frac{1}{\epsilon_{\text{eff}}}]$ that marks the switch from a successful to a failed execution of a circuit. It is possible to represent this by extracting a factor k which results in the following refined metric [SBLW20]:

$$wd < k\frac{1}{\epsilon_{\text{eff}}}$$

As mentioned before, this equation brings width and depth of a circuit in correlation with the error rate. To confirm that this metric holds, we want to benchmark quantum computers with circuits of different sizes. While the error rate of a quantum computer is difficult to quantify precisely because it is dependent on many different factors, the benchmarks are an important step to determine the boundary value of wd [SBLW20].

First, a framework that allows simple benchmarking of quantum computers is created. This framework is implemented in the qiskit-service mentioned in Section 2.3. Since the communication with the IBM backend is already implemented, the focus can be laid on the benchmark itself. To benchmark a quantum computer, we need to create circuits that can be executed. We are using randomized circuits in this thesis which are created using a built-in function of Qiskit [1]. However, their results are not trivially clear as they are simple randomized sequences of gates of given width and depth. To obtain the correct result which we need to judge the quality of the execution on a quantum computer, IBM's quantum simulator *ibmq_qasm_simulator* is used. Since the simulator is not prone to errors caused by decoherence and gate infidelity, it delivers the correct result. While there is an extra amount of effort necessary to retrieve the correct result, the advantage

[1] https://qiskit.org/documentation/stubs/qiskit.circuit.random.random_circuit.html

of randomized circuits is that they are easy to create in large numbers. The number of available implementations of different algorithms is still rather small, hence the approach with randomized circuits was considered the most useful one for this study. With that, large amounts of data can be obtained and analysed.

Having implemented this framework, it is possible to create data by executing various benchmarks. This is done by an API request which allows you to state the target computer, limits for the size of the circuit, the number of circuits you want to create, and the number of shots. Further detail on the implementation will follow in the next chapters.

4.2 Metrics

A statistical analysis of the data is necessary to make an evaluation of the data and quantify the deviation from the quantum simulator's result to the quantum computer's result. Four different metrics were used to judge the similarity of their results. It is possible to interpret the counts as a histogram and use similarity measures to compare the two histograms h_{sim} and h_{real}. Ideally, the metric will quantify the size of the quantum computer's error. An idea is to lay the two histograms on one another and judge the similarity by that. This is where the four metrics come in with different approaches. While the χ^2 distance and the correlation are comparing the general shape the histograms, the percentage error and the histogram intersection are comparing the bars for each count to quantify the error. Both approaches were considered useful to define a success criterion for the benchmarks. A success criterion is needed to be able to judge the quality of the result and classify a benchmark as successful or failed.

4.2.1 χ^2 Distance

The χ^2 *distance* [PW10] is defined as:

$$\chi^2(h_{sim}, h_{real}) = \frac{1}{2}\sum_i \frac{(h_{sim}(i) - h_{real}(i))^2}{h_{sim}(i) + h_{real}(i)}$$

In this equation, h_{sim} is representing the histogram for the simulator and $h_{sim}(i)$ is the simulator's count for bit string i. In the same way, $h_{real}(i)$ represents the quantum computer's count for i. For the χ^2 distance, errors in large bars have a small effect while errors in already small bars will increase the value significantly [PW10].

4.2.2 Correlation

The *correlation* [SSS17] is calculated as follows:

$$corr(h_{sim}, h_{real}) = \frac{\sum_i (h_{sim}(i) - h'_{sim}) \cdot (h_{real}(i) - h'_{real})}{\sqrt{\sum_i (h_{sim}(i) - h'_{sim})^2 \cdot \sum_i (h_{real}(i) - h'_{real})^2}}$$

Like for the χ^2 distance, $h_{sim}(i)$ and $h_{real}(i)$ represent the counts of bit string i for quantum simulator and quantum computer. h'_{sim} and h'_{real} are the sum of all bars for each backend. In this case, they are simply the number of shots indicated in the request. The value of the correlation is normalized between -1 and 1. If the value is close to 1, the histograms are similar whereas values smaller than 0.5 would indicate very strong differences.

4.2.3 Percentage Error

A simple metric that can judge the deviation of the counts from the quantum computer to the correct results from the simulator is the *percentage error* [Hel20]. The percentage error pe of a bit string x that represents a state of a quantum computer is:

$$pe(x) = \frac{|h_{sim}(x) - h_{real}(x)|}{h_{sim}(x)}$$

Here, $h_{sim}(x)$ and $h_{real}(x)$ indicate the number of times x was returned when running the circuit on the quantum simulator, or quantum computer respectively. The percentage error indicates the size of the error a quantum computer made on a specific count in relation to the the correct count. E.g., an error of ten counts has a big impact when the correct count is four. If the correct count is 1000, however, being ten off is a good value. Calculating the percentage error for all correct counts allows to make a judgement on the errors a quantum computer makes during the execution.

4.2.4 Histogram Intersection

The fourth metric that is used to compare the two histograms puts the counts from quantum simulator and quantum computer in relation. It is called *histogram intersection* [SB91] and it compares the two histograms and checks which bars are in both histograms:

$$is(h_{sim}, h_{real}) = \frac{\sum_i \min(h_{sim}(i), h_{real}(i))}{\sum_i h_{sim}(i)}$$

Again, $h_{sim}(i)$ and $h_{real}(i)$ indicate the count for bit string i on the corresponding backend. The denominator is again the same as the number of shots. The histogram intersection can take values from 0 to 1, 0 meaning no similarity and 1 meaning the two histograms are the same.

4.3 Success criterion

With the aforementioned metrics, it is possible to evaluate the quality of a quantum computer's result. However, an arbitrary value does not provide much information without context. It is a difficult step to go from the metric's value to a statement on whether the benchmark should be considered successful or not as it requires to set a boundary on the acceptable error. Saying that only results that are exactly the same as the correct result are successful, is not feasible at this time and would result in every benchmark being considered failed. On the other hand, it would not be useful to accept results that are very erroneous. Evaluating the metrics and deciding on a boundary value is part of Chapter 7. Two exemplary benchmarks are used there to show strengths and weaknesses of the metrics and explain how different degrees of errors affect the values.

Assuming that this provides a valid approach to classify the benchmark as successful or failed, the foundation to determining the maximum circuit size wd is given. The idea is to execute a number of circuits of a specific size, defined by its width and depth. By having a significant sample size for that size, it would be possible to determine whether a quantum computer is capable of executing circuits of the given size successfully or not. This would however require to set a boundary on how many of the benchmarks of size wd need to be successful to consider this specific size feasible on the quantum computer. In this thesis, that boundary was chosen at $\frac{2}{3}$, which was oriented at the heavy output probability [AC16]. As it is not possible to control the size of the transpiled circuit, it is easier to group circuits of similar sizes together, e.g. circuits of $w = 2$ and $d \leq 5$. Starting from that, storing the information about the benchmarks' success rates in a table will allow us to see when the circuit size is too large for the quantum computer because the results become too erroneous. A group of circuits is considered successful if at least two out of three benchmarks are successful. As mentioned before, we can deduce a maximum circuit size for the chosen quantum computer with that information and use that value to calculate the value $\frac{1}{\epsilon_{\text{eff}}}$.

5 Design

We are now taking a closer look at the way the framework was implemented into the qiskit-service. The qiskit-service provides an API that allows the transpilation of Qiskit code to an OpenQASM string which then can be executed on a quantum computer. The algorithm is provided through the Qiskit code, its input parameters, and the desired quantum computer have to be stated in the request. The general workflow of the qiskit-service has been explained in Chapter 2 already. The user sends a request to the API and if necessary a job that will be sent to the IBMQ backend is created with a database entry to store the results once the execution is done. The framework was extended by three new API endpoints that are necessary to run and analyse benchmarks as shown in Figure 5.1. The general workflow has not been changed but instead, the existing functionality is used to execute the benchmarks. We will now take a closer look at the new API endpoints and the general design choices to extend the toolkit.

The first of the new endpoint is `POST /qiskit-service/api/v1.0/randomize` which is used to create and run randomized circuits for benchmarks. As Figure 5.1 shows, the workflow is similar to the execution request which means that after the user sends a request, a database entry is created and

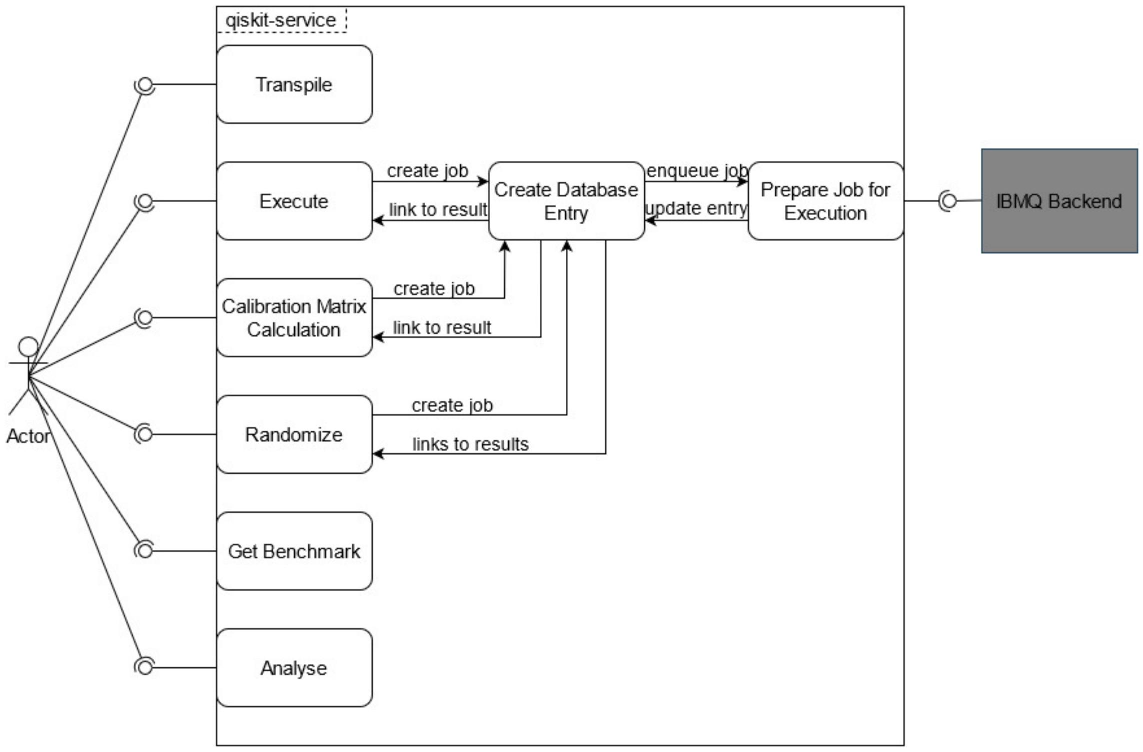
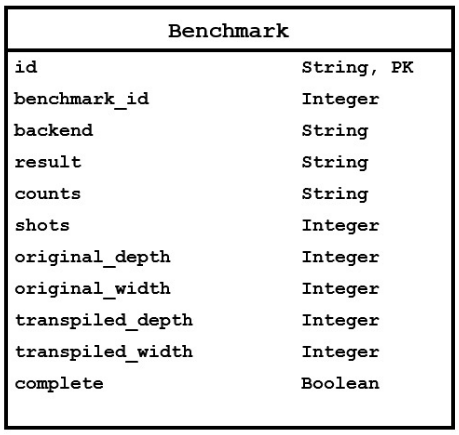


Figure 5.1: Workflow of the updated qiskit-service



Benchmark	
id	String, PK
benchmark_id	Integer
backend	String
result	String
counts	String
shots	Integer
original_depth	Integer
original_width	Integer
transpiled_depth	Integer
transpiled_width	Integer
complete	Boolean

Figure 5.2: Model for Benchmark Entries in the Database

the job is sent to the IBMQ backend. The request body allows the user to adjust the benchmarks, e.g. by different circuit sizes or number of shots. The exact request body will be explained in the next chapter. After the request was sent, randomized circuits of the given size are created using a built-in function of Qiskit. As the correct result is obtained by executing the circuit on the quantum simulator, we need to prepare a job for both the simulator and the chosen backend. Both jobs are saved into a new table in the database. The data model is shown in Figure 5.2. A benchmark entry has an ID field which is an automatically created string and the primary key. Each circuit is tied to a benchmark ID. As already explained, we need to execute each of the randomized circuits on the quantum simulator as well. Since it is not exposed to errors like decoherence or gate infidelity, it can provide optimal results. The benchmark ID allows us to link the result of the simulator and quantum computer when we want to analyse the data and make sure we do not compare the wrong results. The backend field holds the IBM quantum device that was used and the result is the response from the chosen device. For easier access, the counts of the result are saved separately as well as the number of shots. To be able to judge the impact of the circuit size on the success rate, it is necessary to save the circuit size defined by its depth and width as well. The original size is given by the API request, while the transpiled size might differ greatly as the circuit's shape is changed during transpilation. Finally, a boolean is used to indicate whether the job is completed or not. Note that if a job fails to run successfully on IBM's backend, it is still considered completed.

The API endpoint `GET /qiskit-service/api/v1.0/benchmarks/<benchmark_id>` allows users to request the results of a specific benchmark by its ID. It is necessary to state the ID of the benchmark that the user wants to inspect. It gives an overview on the results of the benchmark by providing information on the results on quantum simulator and quantum computer. With that, it is possible to do a manual comparison and rate the quality of the result. As Figure 5.1 shows, the request is handled directly within the API by querying the database and returning the results. It will also be shown in more detail in the next chapter.

Finally, a third API endpoint `GET /qiskit-service/api/v1.0/analysis` is used to run an analysis on the data using the metrics introduced in Chapter 4. The results of the benchmark, both quantum simulator and quantum computer, are taken from the database and then examined. There is no communication needed to IBM's backend as the results are stored in the database. It is important to ensure that the circuits were successfully executed on both backends and the results are correctly saved in the database. Further information on the response, which provides an overview on the values of the metrics to evaluate the quality of the result, is given in the next chapter.

A separate component in the qiskit-service, the *ibmq_handler*, is handling the communication with the IBM backend and ensures that the results can be saved to the database after the execution is finished. This component is also used to send the randomized circuits to the IBMQ backend for the benchmarks used in this thesis. After the job ran on the backend, the ibmq_handler retrieves the result and counts and makes sure these values are saved in the database.

We have seen the fundamental design choices for extending the qiskit-service. A new API endpoint was needed to run benchmarks with randomized circuits of different sizes on IBMQ backends. Two more endpoints are used to either look at one specific benchmark or analyse all available benchmarks using the four metrics introduced in Chapter 4. With that, it is possible to run benchmarks and make a first step to determine the maximum circuit size a specific quantum computer can execute successfully. Having provided a general overview on the design ideas, the next chapter will go into more detail regarding the implementation.

6 Implementation

In this chapter, we take a closer look at the implementation of the framework that was introduced in the preceding chapter. Different technologies were used to create the framework ensuring the functionality. The framework was developed with Python [1] and runs in a Docker container [2]. To create randomized circuits and send them to the IBM backend, Qiskit [3] is used. Flask [4] is used to create the application and manage the requests. The previous chapter already introduced the new API endpoints which will now be explained in greater detail.

Listing 6.1 shows an exemplary body for the POST request which is used to create benchmarks. First, it is necessary to state the quantum computer that should be used for the benchmarks. It needs to be given as the same string that IBM uses in IBM Quantum to name their quantum computers. There are two factors to control the size of the randomized circuit, namely depth and width. The width of the circuit can be simply set by the number of qubits. The depth of the circuit is handled by a lower and an upper limit. In the request body, an interval for the depth of the circuit is stated by specifying a minimum and maximum depth. The framework is implemented so that for each depth within the interval, the given number of circuits is created. Specifying the number of shots is optional and the default value is 1024. Finally, the IBM API token is needed to access IBMQ. The response will include a link to the result of the execution for the quantum simulator and for the quantum computer separately. Additionally, a link to the benchmark is returned which shows a brief summary, including both backends, to the user.

Listing 6.1 Request Body for the `POST` Randomize Request

```
POST /qiskit-service/api/v1.0/randomize
  {
    "qpu-name": "ibmq_athens",
    "number_of_qubits": 2,
    "min_depth_of_field": 1,
    "max_depth_of_field": 5,
    "number_of_circuits": 1,
    "shots": 1024,
    "token": ***
  }
```

[1] https://www.python.org/
[2] https://www.docker.com/
[3] https://qiskit.org/
[4] https://flask.palletsprojects.com/en/2.0.x/

Listing 6.2 Response Layout of the `GET` Benchmark Request with benchmark_id=0

```
GET /qiskit-service/api/v1.0/benchmarks/0
  [
    {
        "backend": "ibmq_qasm_simulator",
        "benchmark_id": 0,
        "complete": true,
        "counts": {
            "000": 1024
        },
        "id": ***,
        "original_depth": 3,
        "original_width": 3,
        "shots": 1024,
        "transpiled_depth": 5,
        "transpiled_width": 3
    },
    {
        "backend": "ibmq_athens",
        "benchmark_id": 0,
        "complete": true,
        "counts": {
            "000": 954,
            "001": 50,
            "100": 20
        },
        "id": ***,
        "original_depth": 3,
        "original_width": 3,
        "shots": 1024,
        "transpiled_depth": 8,
        "transpiled_width": 3
    }
  ]
```

This summary is another API endpoint that was added as part of this thesis and an exemplary response is given in Listing 6.2. As mentioned in Chapter 4, the benchmark ID is used to link the simulator's and quantum computer's result, which provides a direct comparison of the results. By specifying the ID in the request link, an overview on the results of the executions on both backends is provided. This overview includes the backend that was used. The backend is needed to tell which result is the correct one and to show which quantum computer was benchmarked. It also shows the benchmark's ID, a boolean that indicates whether the circuit's execution is finished, and the number of shots for reference. If it is not completed, the response is adjusted to show only that boolean to be false and the IDs of both jobs. Important features are also the original and transpiled size of the circuit and the counts. This allows users to judge the error of the quantum computer manually and put it to relation to the circuit size.

Listing 6.3 Response Layout of the `GET` Analysis Request

```
GET /qiskit-service/api/v1.0/analysis
  [
  ...
    {
        "Benchmark 2" : {
            "Chi Square": 52.567,
            "Correlation": 1,
            "Counts Real": {
                "00": 100,
                "01": 924,
            },
            "Counts Sim": {
                "01": 1024
            },
            "Intersection": 0.9023
            "Percentage Error" : {
                "01": 0.09765625
            },
            "Transpiled Depth": 5,
            "Transpiled Width": 2
        }
    }
  ...
  ]
```

Listing 6.3 shows the response for the request to the last API endpoint. It runs an analysis on all benchmarks that are saved in the database using the metrics that were introduced in Chapter 4, most importantly the histogram intersection. The response contains a list of all benchmarks that were analysed and the values of the metrics for each one. Beyond that, the counts of the simulator's and the quantum computer's result as well as the size of the circuit are given again which provides a broad overview on how the circuit size can affect error rates. To run the analysis, the benchmark table in the database is queried. Only benchmarks where both backends ran the circuit successfully, can be used for the analysis. At this point, it should be noted that for the computation of correlation and intersection, the counts of each backend have to include all results the other backed returned. Consequently, if there is a result the quantum computer returned which is not in the correct result, this result has to be added with count zero, to avoid errors in the application. All values that are needed to calculate the statistical metrics are available in the database. With that information, a brief analysis on the counts can be realized and returned to the user.

Now that the new API endpoints have been covered in detail, we will take a closer look at the creation of circuits and implementation of benchmarks. As mentioned before, it is necessary to run each circuit not only on the quantum computer but also on the quantum simulator, as randomized circuits are used and their results are not given. Given a specific width and depth, random circuits can be created using Qiskit [5]. These circuits have to be transpiled for each backend separately otherwise they cannot be executed. A job needs to be created that can be sent to the IBMQ backend. It will be put into the queue of the backend and after the execution passed, the result is returned.

The qiskit-service already uses an SQLite database to store the results of the execution request. This database was extended by another table to save the benchmarks. The model for the benchmark entries was introduced in Chapter 5. We are using SQLAlchemy [6] to easily manage the database entries. The workflow is similar to the workflow for the execution request that was introduced in Chapter 2. After the API request was sent and the jobs for the IBM backends were generated, database entries for each job are created. The entries are updated once the execution is completed.

We have now covered the foundation of the framework and its design. The framework is used to create data of a large amount of benchmarks. After acquiring data, the framework can be used to analyse it and make an evaluation using the metrics from Chapter 4. The evaluation of the data and conclusions are subject of the next chapter.

[5] https://qiskit.org/documentation/stubs/qiskit.circuit.random.random_circuit.html

[6] https://www.sqlalchemy.org/

7 Evaluation

In this chapter, the data from the benchmarks is analysed to judge the usefulness of the metrics. Furthermore, the effect of the circuit size on the success rate of benchmarks for a specific quantum computer is shown. For the benchmarks in this thesis, IBM's quantum computer *ibmq_athens* was used. It is one of IBM's Falcon processors, has five qubits and a quantum volume of 32 [IBM21].

7.1 Metrics

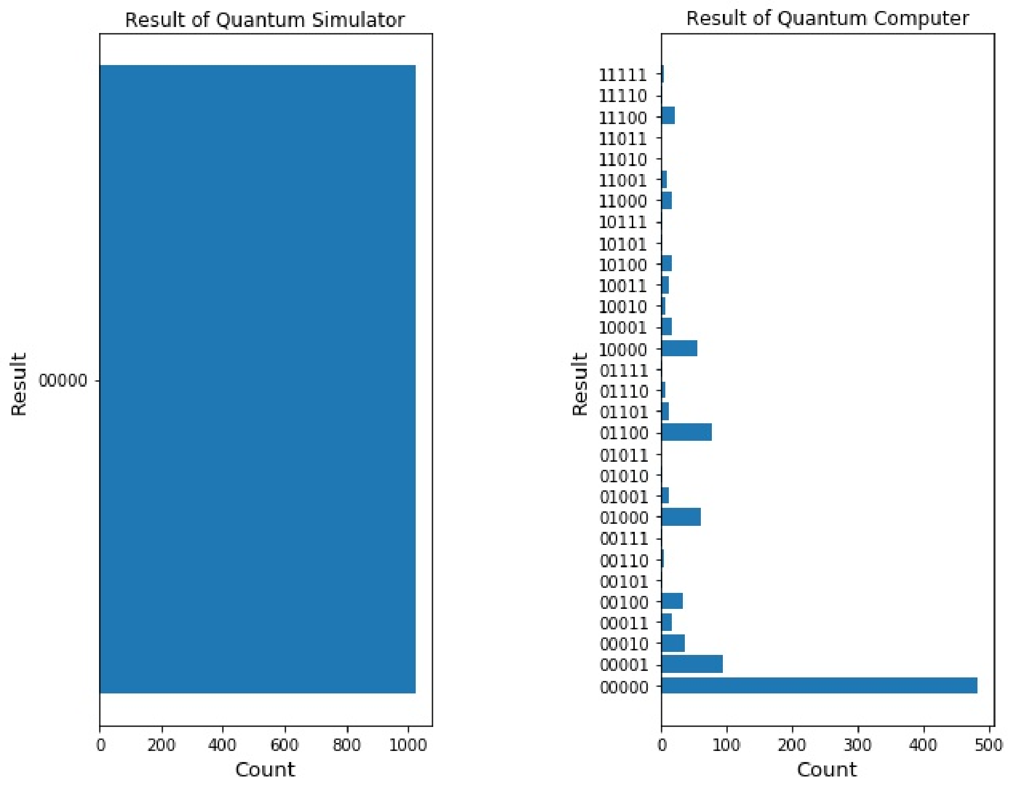
We are now taking a closer look at the metrics introduced in Chapter 4 and use them to evaluate the data. Two exemplary benchmarks with different outcomes were used for the evaluation and to judge the quality of the metrics.

The first benchmark uses a large circuit of depth 52 and width 5. The results of the execution are visualized in the histograms in Figure 7.1. They show a great discrepancy between the correct result from the simulator in Figure 7.1a and the erroneous result from the quantum simulator in Figure 7.1b. Looking at the two histograms, it is easy to see that the quantum computer returned the correct result for less than half of the executions, hence the benchmark should be considered failed.

The second benchmark on the other hand, uses a significantly smaller circuit. Its depth is equal to 6 and its width is equal to 5. The histograms in Figure 7.2 show that the error is much smaller in this case. The histogram illustrates that the difference here is significantly smaller and the benchmark should be considered successful.

Table 7.1 gives an overview on the values the statistical metrics return. With that it is possible to evaluate whether the metrics provide an appropriate measure as to whether a benchmark was executed successfully or not. Looking at the χ^2 distance, we see that it is not a normalized value. Hence, it is not usable to generally verify whether a benchmark was successful or not. Depending on the number of different states in the correct result, the value will vary massively. Because of that, the χ^2 distance is not suitable as a metric for our benchmarks.

Since the former metric is not normalized, the correlation between the two histograms was calculated. The correlation takes a value close to 1 when the two histograms are similar. Similar in this case means that the two histograms indicate a similar distribution of the results. The correlation for both histograms is very close to 1, even though benchmark 1 should be considered failed. The problem here is, that while the results greatly differ, the histogram still looks similar since the bar for 00000 is large while the others are very small. This led to the decision that the correlation is also not applicable as metric to judge the outcome of a benchmark.

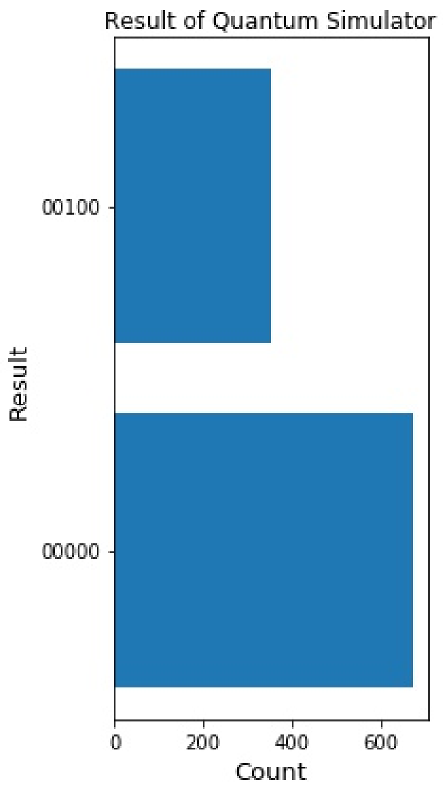


(a) Histogram of Execution on *ibmq_qasm_simulator* **(b)** Histogram of Execution on *ibmq_athens*

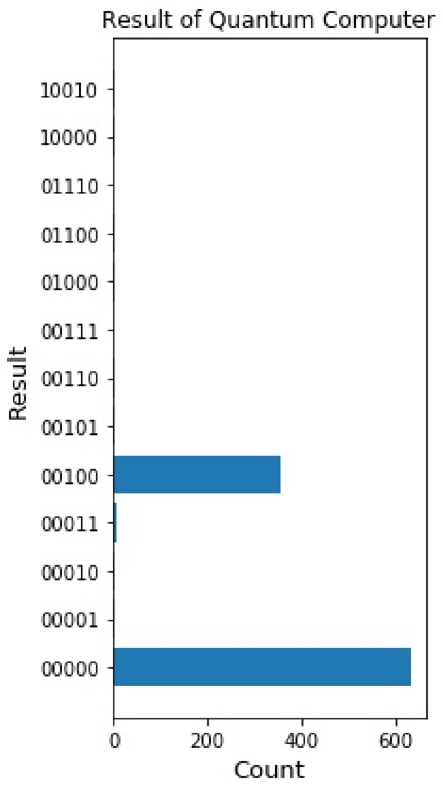
Figure 7.1: Histograms for Benchmark 1

The percentage error however, proves to give a better estimate on the error of the execution. For benchmark 1, it is slightly above 50% which indicates that there is a big error and the benchmark should be considered failed. Benchmark 2 on the other hand, has errors that are smaller than 10% which is only a small deviation. However, due to the fact that the percentage error has to be calculated for each count separately, it does not provide a general view on the result. While it would be possible to take the biggest calculated percentage error value as reference, the histogram intersection proved to be the better choice as it returns just one value that directly can be used to decide on the outcome of the benchmark.

The histogram intersection is normalized between 0 and 1 where 1 indicates that two histograms are the same, while 0 indicates there are no similarities. For the first benchmark the histogram intersection is at 0.4717. This means that less than 50% of the two histograms overlap, hence the result can be considered very erroneous. On the other hand, it is almost 96% for benchmark 2 which is a very good value. Since the histogram intersection indicates how much of the histograms overlap, it is a good measure to judge whether a benchmark ran successfully or not. It only requires the user to set a boundary value. Here, an error of 10% is considered acceptable, meaning that a benchmark is considered successful if the histogram intersection is greater than 0.9. This value was chosen by manually comparing the results of benchmarks. Picking a value closer to 1 would result in most benchmarks being considered failed, as the error rates are still very much noticeable. On



(a) Histogram of Execution on *ibmq_qasm_simulator*



(b) Histogram of Execution on on *ibmq_athens*

Figure 7.2: Histograms for Benchmark 2

	Size		**Statistical metrics**			
Benchmark	w	d	χ^2	$corr$	pe	is
1	5	52	367.6072	0.9613	00000 : 0.5283	0.4717
2	5	6	18.7018	0.9935	00000 : 0.0624 00100 : 0.0171	0.9590

Table 7.1: Two exemplary benchmarks on *ibmq_athens* with transpiled circuit width w and circuit depth d on the quantum computer analysed with the following statistical metrics: χ^2 distance χ^2, correlation $corr$, percentage error (per count) pe and intersection is

the other hand, using a smaller value would result in accepting results that show a great deviation from the correct result. This is why 0.9 is considered a reasonable boundary value to classify a benchmark as failed or successful.

7.2 The Effect of Circuit Size on Success Rate

In this section, the data from the benchmarks is used in an attempt to determine a maximum circuit size for a specific quantum computer, in this case *ibmq_athens*. The success rates of the benchmarks have been put in Table 7.2. As the size of the transpiled circuit, especially the depth, cannot be controlled, circuits of similar sizes were grouped together. By grouping the circuits according to their depth, e.g. all circuits of $d \leq 5$, it is easy to get a clear overview on the success rates. These groups are called classes from here on. One class includes all circuits of a specific width and of a specific range for the depth, like $d \leq 5$. As the chosen quantum computer *ibmq_athens* has five qubits, the maximum width is also five. Entries are structured x/y which means that from y benchmarks, x were successful. Entries marked as *None* indicate that no circuit of the given size could be created. In Chapter 4 a success criterion was introduced: If at least two out of three benchmarks of a specific class are successful, that class is considered successful as a whole.

Taking a closer look at the values for circuits of width 1, we can see that smaller circuits up to a depth of 20 run successfully most of the time. There is a significant spike in the error rate above that with only 11 out of 18 benchmarks being successful when the circuit size is greater than 20. However, it should be noted that the amount of circuits is relatively small for that group. Most of the circuits with width 1 were transpiled to rather small circuits no matter how deep the original randomized circuit was. The available data suggests that for circuits of depth 1, anything up to a depth of 20 is expected to run successfully but beyond that, the results would be too erroneous. While the table shows that 4 out of 6 benchmarks for circuits with a depth greater than 30 were successful, the sample size is very small. Additionally, the success rate for circuits with $d > 20$ is already too small. Hence, the class of $d > 30$ is considered failed as well. Taking a look at the rule of thumb metric, this suggests that wd should be ≤ 20.

The data for circuits of width 2 is very limited, as most circuits were transpiled to have a depth smaller than 10. The depth of the randomized circuits could not be increased arbitrarily as that causes the simulator to fail execution. Without the simulator's result it is not possible to judge the quality of the result. Table 7.2 shows that anything up to depth 10 is expected to run successfully. Beyond that, it is not possible to make a valid prediction. For the rule of thumb metric this supports $wd \leq 20$ again, however it does not give any information on whether larger circuits could run successfully.

Moving on to circuits of width 3, we see that circuits with a depth of up to 10 are expected to run with strong confidence. In the interval from 11 to 15, a confident prediction is not possible as the sample size is too small and circuits with a greater depth than 15 are expected to deliver very erroneous results and hence should not be executed. In this case, we predict an acceptable result as long as $wd \leq 30$.

At width 4, we can see that the success rate declines very fast. While it is acceptable for circuits of depth 10 or smaller, anything larger than that is expected to fail badly. The success rate for circuits of depth 6 to 10 is made from a small sample of 11 and should be treated with caution. However, the data for these circuits strongly supports $wd \leq 20$ again. Circuits of size 30 to 40 can be expected to run successfully too, but the confidence should not be as high as for smaller circuits.

Randomized circuits of width 5 result in circuits of great depth during the transpilation. Hence, there is a relatively small sample of small circuits but it suggests a similar outcome as the data for circuits of smaller width. Here, circuits that have a smaller depth than 10 are expected to run

	$w = 1$	$w = 2$	$w = 3$	$w = 4$	$w = 5$
$d \leq 5$	64/64	44/44	48/48	30/34	13/15
$5 < d \leq 10$	33/33	121/125	16/17	8/11	6/7
$10 < d \leq 15$	28/31	None	3/4	0/3	1/8
$15 < d \leq 20$	21/25	None	1/17	1/26	0/20
$20 < d \leq 25$	7/12	None	0/17	0/15	0/27
$25 < d \leq 30$	None	None	0/2	0/6	0/9
$d > 30$	4/6	None	0/18	0/18	0/15

Table 7.2: Success rates of benchmarks on *ibmq_athens* by the size of transpiled circuits

successfully again. For circuits using 5 qubits we therefore accept anything with a depth ≤ 5. As the data set for $5 < d \leq 10$ is very small, accepting it comes with a risk. The data shows that the success rates drops significantly again when the circuit's depth is greater than 10 which indicates that the circuit size massively influences the success rate. The data set for width 5 suggests that $wd \leq 25$ is definitely acceptable and even values up to 50 can deliver good results but those should be treated with caution.

The data of all sizes suggests that randomized circuits of sizes up to 20 can be executed confidently while circuits of sizes between 25 and 40 will introduce small errors. Larger circuits should be avoided as they most likely will return very erroneous results.

7.3 Discussion

This thesis is a step to benchmark quantum computers to compute the maximum circuit size that a specific quantum computer can successfully handle. However, there are limitations to the provided framework which will be discussed in this section.

First, this framework uses only randomized circuits. Randomized circuits are easy to create but they do not necessarily represent real applications and it is not guaranteed that the same assumptions hold for non-randomized circuits. However, by executing a large amount of randomized circuits, it is possible to increase the confidence in the results as a wide range of different circuits will be tested. While existing implementations of algorithms [1] were used and tested in the beginning of this thesis, the framework does not support running them automatically. A manual analysis of that data proved to be difficult as it was not saved in a database with necessary information like circuit size and counts easily accessible. The approach to use randomized circuits proved to be easy and efficient to implement.

The sample size is another limitation here, which was mentioned before. As it is not possible to control the size of the transpiled circuit, it is difficult to get a large sample size for one specific circuit size. Hence, several of the success rates from before need to be treated carefully because with a small sample size of two or three, it is still very much possible that only exceptions were

[1] https://github.com/UST-QuAntiL/nisq-analyzer-content/tree/master/example-implementations

observed and the sample is not statistically significant. Despite the limited amount of data, the trend seems to be confirmed. There is no irregular observation that suggests circuits of size $wd \leq 20$ are already a problem for *imbq_athens*, but instead the data from the different benchmarks is piled up around that value. Nonetheless, filling the gaps in Table 7.2 and having a bigger sample size will help with confirming that this assumption really holds.

Furthermore, a very sharp cut is made with the intersection metric at a value of 0.9 when it is used to evaluate the benchmarks. If the histogram intersection is less than 0.9, the execution is considered failed as it indicates a significant difference between the two histograms. While it does offer a first step to judge the quality of a result, it is not ideal. The acceptable error is also dependent on the application. If quantum computers can be used for e.g. security related applications in the future, precision will be very important and a value of 0.9 is likely not good enough. It would be interesting to take the exact value of the intersection metric and use the data to not only predict if a circuit will run successfully or not, but predict the value of the histogram intersection for unknown circuits. With that, it would be possible to quantify the error of a given circuit by its size. It would be a more general approach that would not be based on a strict boundary value.

In this thesis, only four different metrics were used to compare the results of quantum simulator and quantum computer. While there are more metrics available, in this case the histogram intersection proved to be useful and provided a good measure on the similarity of the two results.

8 Conclusion and Outlook

In this thesis, a framework to easily benchmark quantum computers with randomized circuits of certain size is provided. It creates circuits to execute them on IBM's quantum simulator and on the chosen quantum computer and offers an analysis of the collected data. Using the intersection metric, it is possible to compare the histogram of the quantum computer's result with the histogram of the quantum simulator's result. The quantum simulator returns the correct result and if the intersection metric returns a value greater than 0.9, the benchmark is considered successful. Correlation and χ^2 distance do not provide a normalized and easy to use metric and were therefore not deemed applicable to judge whether a benchmark ran successfully or not. While the correlation often is close to 1 even for very erroneous results, the χ^2 distance can be any positive number and therefore cannot be used to compare the results of different benchmarks. For that, we need a normalized metric that measures the error of the result. The percentage error is normalized and provides a good metric to judge the quality for each count separately, but it does not provide a validation whether the result as a whole is acceptable. Hence, the decision was made to use the intersection metric that states how much of the ideal histogram of the quantum simulator's result is covered by the histogram of the quantum computer's result.

With the aforementioned framework, the effect of the circuit size on the success rate can be studied. We learned that for IBM's quantum computer *imbq_athens*, circuits of size 20 are expected to run successfully. This means that a circuit of width 1 can be very deep and still deliver good results, while a circuit of width 4 or 5 will return very erroneous results once the depth is greater than 10. However, the circuit size wd is still larger for those wide circuits as it goes up to 40 or even 50 with acceptable results. It was not feasible to create circuits for specific circuit sizes, as the actual size of the transpiled circuit cannot be controlled at this point in time but an overall view on the data suggests this trend. This value can be used to validate the sharpened rule of thumb metric $wd < k\frac{1}{\epsilon_{\text{eff}}}$. With that, the framework provides a good foundation to determine the value $\frac{1}{\epsilon_{\text{eff}}}$ for a specific quantum computer.

Outlook

Going beyond what was covered in this thesis, it would be interesting to have a definite iterative way to increase the size of the transpiled circuit, which would allow a more precise analysis of the data. If it were possible to control the size of the transpiled circuit, the next step would be creating large numbers of circuits of each size. This would allow a prediction with much greater confidence than in this thesis as a sample size of e.g. 100 is statistical more significant. Furthermore, with more data available, a machine learning approach becomes interesting. The data that is available through the analysis can be used for learning to predict the error rate for an unknown circuit of given size.

As mentioned before, the framework to execute benchmarks that is provided in this thesis, is very general. So far, the framework works only with randomized circuits. Extending it to include implementations of known algorithms would allow further analysis on whether the type of the circuit affects the error of the result as well. Another possible next step is to extend the framework further to take the quantum computer's error rate into account. With that, it would be possible to validate the refined metric $wd < k\frac{1}{\epsilon_{\text{eff}}}$. Having a precise prediction on whether a quantum circuit is executable on a specific quantum computer or not will be immensely helpful when automatically selecting a quantum computer for a specific quantum algorithm. This will be an important step to improve the accessibility of quantum computers.

Bibliography

[AAA+19] H. Abraham et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: 10.5281/zenodo.2562110 (cit. on p. 21).

[AC16] S. Aaronson, L. Chen. *Complexity-Theoretic Foundations of Quantum Supremacy Experiments*. 2016. arXiv: 1612.05903 [quant-ph] (cit. on p. 30).

[Bea03] S. Beauregard. "Circuit for Shor's Algorithm Using 2n+3 Qubits". In: 3.2 (Mar. 2003), pp. 175–185. ISSN: 1533-7146 (cit. on p. 18).

[BY20] R. Blume-Kohout, K. C. Young. *A volumetric framework for quantum computer benchmarks*. 2020. eprint: 1904.05546 (cit. on p. 24).

[CBS+19] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, J. M. Gambetta. "Validating quantum computers using randomized model circuits". In: *Physical Review A* 100.3 (Sept. 2019), p. 032328. DOI: 10.1103/PhysRevA.100.032328 (cit. on p. 23).

[CBSG17] A. W. Cross, L. S. Bishop, J. A. Smolin, J. M. Gambetta. *Open Quantum Assembly Language*. 2017. arXiv: 1707.03429 [quant-ph] (cit. on p. 21).

[Hel20] A. M. Helmenstine. *How to Calculate Percent Error*. https://www.thoughtco.com/how-to-calculate-percent-error-609584. Nov. 2020 (cit. on p. 29).

[IBM21] IBM. *IBM Quantum*. https://quantum-computing.ibm.com. 2021 (cit. on pp. 19, 21, 25, 39).

[Ker19] S. Kersken. *IT-Handbuch für Fachinformatiker*. Rheinwerk Computing, 2019 (cit. on p. 17).

[LB20] F. Leymann, J. Barzen. "The bitter truth about gate-based quantum algorithms in the NISQ era". In: *Quantum Science and Technology* (Sept. 2020), pp. 1–28. DOI: 10.1088/2058-9565/abae7d (cit. on pp. 15, 20, 23, 24).

[MBB+18] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn, et al. "Quantum optimization using variational algorithms on near-term quantum devices". In: *Quantum Science and Technology* 3.3 (June 2018), p. 030503. DOI: 10.1088/2058-9565/aab822 (cit. on pp. 15, 23).

[MN19] A. Matuschak, M. A. Nielsen. *Quantum Computing for the Very Curious*. https://quantum.country/qcvc. San Francisco, 2019 (cit. on pp. 17–19).

[MNW+17] K. Michielsen, M. Nocon, D. Willsch, F. Jin, T. Lippert, H. De Raedt. "Benchmarking gate-based quantum computers". In: *Computer Physics Communications* 220 (2017), pp. 44–55. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2017.06.011 (cit. on pp. 18, 21, 24).

[NC10] M. A. Nielsen, I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010 (cit. on pp. 17, 18, 20).

[Pre18] J. Preskill. “Quantum Computing in the NISQ era and beyond”. In: *Quantum* 2 (Aug. 2018), p. 79. DOI: 10.22331/q-2018-08-06-79 (cit. on pp. 15, 20, 23, 24).

[PW10] O. Pele, M. Werman. “The Quadratic-Chi Histogram Distance Family”. In: vol. 6312. July 2010, pp. 749–762. ISBN: 978-3-642-15551-2. DOI: 10.1007/978-3-642-15552-9_54 (cit. on p. 28).

[RP11] E. Rieffel, W. Polak. *Quantum Computing - A Gentle Introduction*. The MIT Press, 2011 (cit. on pp. 17, 18, 20).

[SB91] M. J. Swain, D. H. Ballard. “Color Indexing”. In: *Int. J. Comput. Vision* 7.1 (Nov. 1991), pp. 11–32. ISSN: 0920-5691. DOI: 10.1007/BF00130487 (cit. on p. 29).

[SBB+20] M. Salm, J. Barzen, U. Breitenbücher, F. Leymann, B. Weder, K. Wild. “The NISQ Analyzer: Automating the Selection of Quantum Computers for Quantum Algorithms”. In: *Communications in Computer and Information Science* (Nov. 2020), pp. 66–85. DOI: 10.1007/978-3-030-64846-6_5 (cit. on pp. 15, 20, 21, 24, 25).

[SBLW20] M. Salm, J. Barzen, F. Leymann, B. Weder. “About a Criterion of Successfully Executing a Circuit in the NISQ Era: What $wd \ll 1/\epsilon_{\text{eff}}$ Really Means”. In: *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software*. Association for Computing Machinery, Nov. 2020, pp. 10–13. DOI: 10.1145/3412451.3428498 (cit. on pp. 15, 23–25, 27).

[SDC+20] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, R. Duncan. “t|ket⟩: a retargetable compiler for NISQ devices”. In: *Quantum Science and Technology* 6.1 (Nov. 2020), p. 014003. DOI: 10.1088/2058-9565/ab8e92 (cit. on p. 25).

[SSS17] Y. Shikhar, V. Singh, R. Srivastava. “Comparative Analysis of Distance Metrics for Designing an Effective Content-based Image Retrieval System Using Colour and Texture Features”. In: *International Journal of Image, Graphics and Signal Processing* 9 (Dec. 2017), pp. 58–65. DOI: 10.5815/ijigsp.2017.12.07 (cit. on p. 28).

[SZR16] E. A. Sete, W. J. Zeng, C. T. Rigetti. “A functional architecture for scalable quantum computing”. In: *2016 IEEE International Conference on Rebooting Computing (ICRC)*. 2016, pp. 1–6. DOI: 10.1109/ICRC.2016.7738703 (cit. on pp. 15, 23).

All links were last followed on May 24, 2021.

Declaration