Testing the assumptions of recent quantum supremacy experiments

Wenxing Duan

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2023

Abstract

In this dissertation, I primarily concentrate on analyzing Google's quantum random circuit sampling experiment, with the objective of evaluating their quantum supremacy claims. I employed Fourier analysis as the method to investigate the relationship between the experimental results and the number of qubits, allowing for a comprehensive assessment of the strength of Google's quantum supremacy assertion. During the course of my research, I encountered an intriguing phenomenon. Contrary to expectations, my experimental findings revealed that Google's experiment exhibited higher correlator values than those of a theoretically noise-free ideal circuit. This discrepancy prompted an extensive analysis of the data to identify the underlying cause, which was eventually attributed to an insufficient number of samples. Moreover, the level of noise in the system was found to significantly influence the outcomes associated with varying sample sizes. Consequently, determining the optimal sample size required to obtain correlators close to their true values has emerged as a valuable research question, warranting further investigation.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Wenxing Duan*)

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Raul Garcia-Patron Sanchez. Throughout my entire thesis, he provided meticulous guidance accompanied by immense patience and wisdom. Dr. Sanchez's encouragement and support have been an indispensable part of my journey in completing this thesis.

Next, I would like to thank my parents, Yongqiang Duan and Dongmei Feng, who have worked tirelessly day and night to provide the essential financial support for my four years of university. I would also like to express my gratitude to my late grandmother, Xiulan Zhu, who always encouraged me to bravely pursue my dreams and served as a significant source of emotional strength throughout my studies.

Finally, I would like to extend my appreciation to all my friends who have been by my side. I am particularly grateful to Yanchen Fan, Yuyang Xin, and Jingwen Li for patiently listening to my complaints and grievances. I would like to thank Ke Shen and Yiming Luo for offering valuable advice during my 4th year. I appreciate Lucy Lin for encouraging me and providing emotional support during my most challenging moments. I would also like to thank all members in chat group "Cool Cooler Coolest" for accompanying me through every late-night work session.

I am truly grateful for each and every one of you who have supported and encouraged me throughout this journey. Your unwavering belief in me has made all the difference, and I could not have accomplished this without you all. Thank you for being a part of my academic and personal growth.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

With the development of computer science, it has been found that more and more problems are difficult to solve in a short time on a classical computer architecture. NP problems, which cannot be solved efficiently by classical computers in polynomial time, have become increasingly relevant in many fields, including cryptography, optimization, and chemistry. Fortunately, quantum computing, an emerging computing architecture, offers promising potential for accelerating the resolution of these complex problems. Quantum computing is based on quantum mechanics, using the superposition effect of qubits, enabling quantum computers to perform computations exponentially faster than classical computers for some of the problems.

Quantum computing has been a topic of study for over 40 years, and in that time, scholars have established a solid theoretical foundation for quantum computing. Researchers have proposed many algorithms to solve practical problems, such as Shor's algorithm, which could be used to factoring large numbers and Grover's algorithm for searching databases. Moreover, quantum computing has had a significant impact on other fields, such as cryptography, where it has been shown that quantum computers can break some of the most commonly used encryption schemes such as RSA.

However, the question of whether quantum computers can truly solve problems that are infeasible for classical computers in a practical amount of time remains a topic of debate. In 2019, Google claimed to have achieved quantum supremacy using their 53-qubit quantum computer, "Sycamore" [1]. This claim was soon challenged, as subsequent analysis revealed that the calculation in question would take only a few days on a classical system [2].

1.2 My Project

In this dissertation, I will investigate the assumption of Google's claim of quantum supremacy by applying the Fourier analysis on the data generated by Sycamore .Fourier analysis on Boolean functions is a mathematical technique used to decompose and study these functions in terms of their orthogonal basis, derived from the Walsh-Hadamard basis, revealing patterns and properties that may not be apparent in their original representation. By applying Fourier analysis to the sample data generated by Sycamore, I will be able to calculate the Fourier coefficients of the output samples, which represent the correlations among the different qubit measurement at output. This study will help determine how strong the assumption of supremacy of Sycamore is and contribute to the ongoing debate on the potential of quantum computing.

Overall, quantum computing has the potential to revolutionize many fields, and testing its capabilities is crucial to realizing this potential. By investigating Google's claim of quantum supremacy, this dissertation aims to contribute to the advancement of quantum computing and its applications in various fields.

1.3 Structure

Chapter 2: In this chapter, I will provide an overview of quantum computing, including its historical and background knowledge. I will also explore the idea of quantum supremacy. Specifically, I will focus on the Quantum Random Circuit Sampling (RCS) experiment conducted by Google on its 53-qubit quantum computer "Sycamore" in 2019. I will describe the RCS process in detail and discuss its significance in the field of quantum computing. By examining the RCS experiment and related research, this chapter aims to provide a basic understanding of quantum supremacy and RCS experiment.

Chapter 3: In this chapter, I will look into the mathematics behind the Fourier analysis of the real-valued Boolean functions of probability distributions. Specifically, I will explain the step-by-step process of the mathematical formulas used in Fourier analysis and their purpose in analyzing the correlation between input qubits and output samples. By providing a detailed analysis of the mathematical formula of Fourier analysis, this chapter aims to enhance the reader's understanding of the analysis process and its purpose of the investigation of quantum supremacy.

Chapter 4: In this chapter, I'll detail the Python implementation of the function, providing the detail of the code, using optimization techniques and GPU computing to achieve a 3800x speed increase in Fourier coefficient computation. I'll also discuss the validation of my results.

Chapter 5: In this chapter, I present a comprehensive analysis of Google's random circuit sampling results obtained from the Sycamore quantum processor. The primary focus of our investigation lies in examining both the Patch circuit and Full circuit configurations independently across varying qubit sizes, ranging from 12 to 24 qubits. By conducting a detailed comparative analysis of the results, I aim to uncover the impact

of qubit count on the accuracy of quantum computations and explore the strengths and limitations of the Sycamore processor.

Chapter 6: In this final chapter, I will present a thorough recap of the dissertation. This summary will encapsulate the main objectives, methods, outcomes, and implications of the research explored in this work. By providing a concise overview, readers will be able to quickly grasp the key aspects and findings discussed throughout the dissertation.

Chapter 2

Background

2.1 Overview

Quantum computing is an innovative technology that leverages the principles of quantum mechanics, such as superposition and entanglement, to address problems that are challenging for classical computers to solve. Classical computers operate with bits that are either 0 or 1, utilizing logic gate circuits to execute operations and ultimately producing deterministic results. In contrast, quantum computers employ qubits, which represent a superposition of 0 and 1. This means that a quantum computer with n qubits can simultaneously perform operations on 2^n inputs. This computational paradigm exhibits immense potential in diverse fields, including quantum system simulation, computational biology, machine learning, and search problems. Furthermore, its ability to rapidly factorize large integers has significant ramifications for the field of cryptography. Researchers refer to a quantum computing device's capability to solve problems that classical computers cannot feasibly address within a reasonable time frame as "quantum advantage."

Ever since Feynman first introduced the concept of quantum computing in 1981, the field has experienced remarkable growth. Major technology companies, such as Google, Microsoft, and IBM, have dedicated substantial resources to investigating the vast potential of quantum computing. In this chapter, we will delve into the milestones of quantum computing development, shedding light on the pursuit of quantum supremacy and highlighting the most recent research breakthroughs.

The pursuit of quantum computing technology has led to the development of various quantum computing architectures, such as superconducting qubits, trapped ions, and photonic systems, each with its own set of advantages and challenges. As researchers continue to refine these architectures, they also explore novel error-correction techniques to ensure the stability and reliability of quantum computations.

In parallel with hardware developments, theoretical advancements in quantum algorithms and programming languages have driven the field forward, allowing for more efficient problem-solving using quantum computers. These advancements have been

instrumental in demonstrating the potential of quantum computers to revolutionize fields such as drug discovery, materials science, and financial modeling.

As the field of quantum computing continues to evolve, so does the understanding of its implications for society, including the potential risks and ethical considerations. The ongoing dialogue among researchers, policymakers, and industry leaders ensures that this transformative technology is developed responsibly and used for the betterment of humanity.

2.2 Quantum Computing

2.2.1 Idea Proposing Stage

In 1981, Richard Feynman first proposed the idea of a quantum computer during a conference [3]. He envisioned a computing device distinct from traditional Turing machines, capable of simulating complex quantum physical systems that were difficult for conventional computing devices. Inspired by Feynman's ideas, David Deutsch formally defined a quantum computer in 1985 [4]. Since then, scientists have been exploring areas where quantum computers have advantages over conventional computers.

2.2.2 Theory Matures Stage

In 1992, David Deutsch and Richard Jozsa jointly proposed the world's first quantum algorithm, known as the Deutsch–Jozsa algorithm [5]. A couple of years later, in 1994, Daniel Simon developed the Simon algorithm while solving the Simon problem [6]. This directly inspired Peter Shor to design the Shor algorithm, which solved the prime factorization problem of large integers [7]. Shortly thereafter, in 1996, the invention of Grover's algorithm [8] also considered the second major quantum computing algorithm after Shor's algorithm, which marked a new phase in human exploration of quantum computing advantages.

2.2.3 Actual Quantum Computer Building Stage

Until 1997, quantum computing remained mostly theoretical. However, that year, Isaac L. Chuang and other researchers introduced a quantum machine design based on nuclear magnetic resonance (NMR) machines, which they built in 1998 [9]. In 1999, Yasunobu Nakamura and Jaw-Shen Tsai proposed the idea of building quantum computers using superconducting devices [10]. The 2-bit quantum computer they constructed implemented Grover's algorithm, and soon after, in 2001, IBM built a 7-bit quantum computer that implemented Shor's algorithm, successfully factoring 15 into 3 and 5 [11]. In 2003, the Deutsch–Jozsa algorithm was implemented on an ion-trap quantum computer [12]. By 2006, researchers at the University of Waterloo successfully built a 12-qubit quantum machine [13]. A year later, the C-NOT gate was successfully developed on a superconducting quantum machine [14].

In 2009, researchers from the University of Bristol successfully ran Shor's algorithm on a photonic chip [15]. In 2011, D-Wave developed the first commercial quantum computer, utilizing quantum annealing technology, while scholars produced a quantum machine with a von Neumann architecture on a superconducting circuit [16]. In 2015, D-Wave claimed to have broken the 1000-qubit barrier. A year later, MIT scientists implemented Shor's algorithm in an ion-trap quantum computer [17]. In 2017, IBM demonstrated a 50-qubit computer that could maintain its quantum state for 90 microseconds. In 2019, Google claimed to have achieved quantum supremacy through random circuit sampling on their superconducting quantum computer "Sycamore" [1]. More details about quantum supremacy will be provided in the next section.

2.3 Quantum Supremacy

Quantum supremacy, or quantum advantage, is a pivotal milestone in quantum computing, marking the moment when a quantum computer can solve a problem significantly faster or more efficiently than the best available classical computer. This concept underscores the practical potential of quantum computers to outperform classical computers in specific tasks. The term "quantum supremacy" was firstly mentioned by J. Preskill in 2012 [18]., aiming to identify a computational problem where a quantum computer provides a clear advantage. Achieving quantum supremacy requires a scalable quantum computer, an effective quantum algorithm, and the ability to verify results using classical computers. The main way to verify quantum supremacy is by building a computational framework that is difficult for classical computers to handle but easy to process on a quantum computer. However, quantum supremacy is typically attainable only under ideal quantum circuit conditions. As real-world quantum circuits are prone to noise, the final outcomes may deviate from the expected results. Today, researchers mainly verify quantum supremacy through two kinds of experiments.

In 2019, Google claimed to have reached quantum supremacy using their 53-qubit processor called "Sycamore." [19][1] They reported solving a problem related to random circuit sampling [20] much faster than the world's leading classical supercomputer at the time. This claim sparked considerable interest and debate within the scientific community.

In 2020, researchers from the University of Science and Technology of China (USTC) reported achieving quantum supremacy using a different approach called "Gaussian boson sampling" [21] with their photonic quantum computer, Jiuzhang [22]. This experiment further demonstrated the potential of quantum computing and provided an alternative method to showcase quantum advantage. In 2022, Xanadu conducted a larger scale Gaussian boson sampling experiment on the Borealis photonic processor [23]. Utilizing up to 219 photons, with an average of 125 photons, they claimed to achieve results that would require 9,000 years for a classical computer to accomplish.

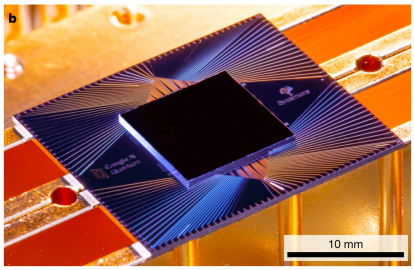


Figure 2.1: Sycamore Superconduction Quantum Computer [1]

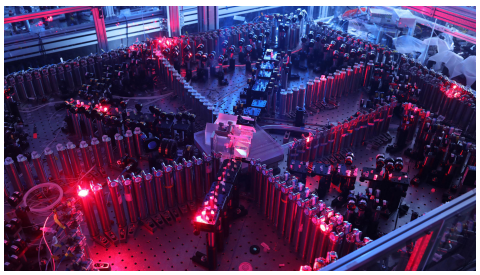


Figure 2.2: Jiuzhang Photonic Quantum Computer [22]

Besides, Moreover, alternative approaches for demonstrating quantum supremacy are also available, such as Shor's algorithm, D-Wave's specialized frustrated cluster loop problems [24], etc. In this dissertation, I will focus on the random circuit sampling experiment done by Google.

2.3.1 Random Circuit Sampling

Random Circuit Sampling (RCS) is a process that is hard for classical computer to simulate. In RCS, a random quantum circuit is created by applying a series of quantum gates to a set of qubits in a specific order. The gates are chosen randomly from a predefined set, such as the universal gate set, which includes single-qubit gates like the Pauli-X, Pauli-Y, Pauli-Z, Hadamard, and T-gates, and two-qubit gates like the controlled-NOT (CNOT) gate. The random circuit generated represents a complex unitary transformation that manipulates the states of the qubits.

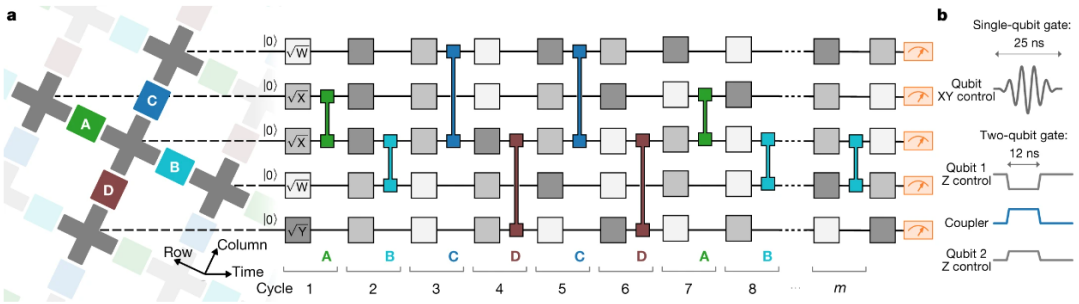


Figure 2.3: A example quantum circuit instance used in Google experiment [1].

After the quantum circuit is executed, the final state of the qubits is measured, collapsing the qubits from their superposition states into classical bit values (0 or 1). The measurement outcomes represent a sample from the probability distribution produced by the random quantum circuit. Repeating the process multiple times with different random circuits generates a larger set of samples with n bits length from the corresponding probability distributions, where n represent the qubit size.

The RCS problem essentially consists of sampling from these probability distributions generated by random quantum circuits. While quantum computers can perform

this task relatively efficiently by directly implementing the random quantum circuits, classical computers struggle to simulate the process due to the exponentially growing complexity of the quantum states as the number of qubits and gates increases.

But in the actual quantum computer, the noise will significantly affect the output result. Reflected in the results, the results of the measurement will be biased towards some specific output. To quantify the fidelity of quantum computers and quantum circuits, the cross-entropy benchmark was used in this paper. The probability of a bitstring in ideal quantum can be simulated and computed on classical machine. And by analyzing the large number of experimental data coming out from the quantum computer, the probability of a certain output can be computed. Then comparing the ideal probability with real probability, High Output Generation (HOG) can be calculate.

$$HOG(p,q) = \sum_x q(x)p(x)$$

Where fidelity can be expressed in term of:

$$\mathcal{F}_{XEB} = 2^n \langle P(x) \rangle_x - 1$$
$$\mathcal{F}_{XEB} = 2^n HOG - 1$$

In the ideal noiseless situation, $\mathcal{F}_{XEB}$ shoule be equal to 1, and in the case of full noise, $\mathcal{F}_{XEB}$ should be close to 0. According to Google experiment [1], $\mathcal{F}_{XEB}$ decrease exponentially with the increase of the qubit number.

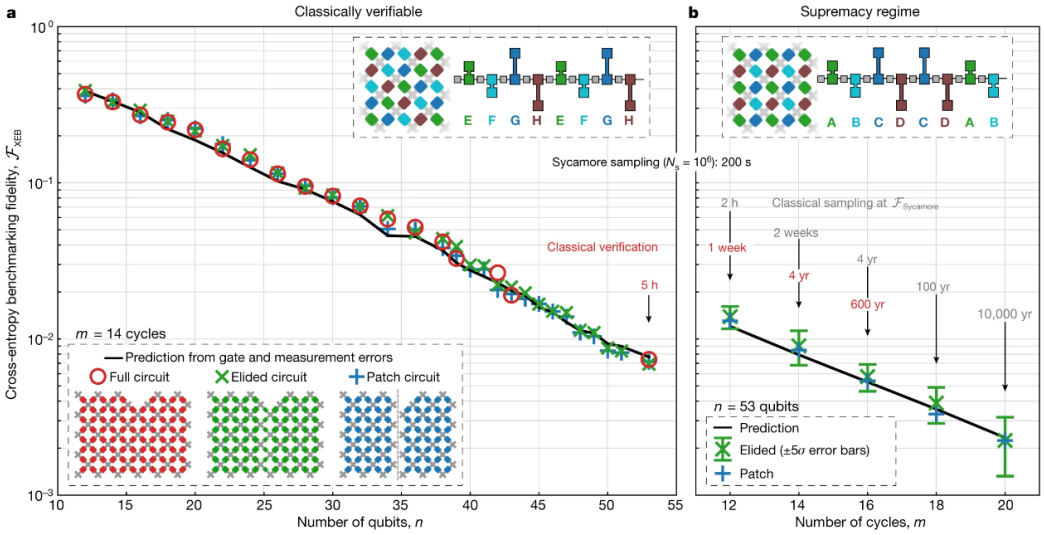


Figure 2.4: Google Experiment [1]

In this experiment, Google conduct 3 different kinds of experiments, which are **Full circuit**, **Elided circuit** and **Patch circuit**. These three different circuit indicate the complexity of the circuit. The full circuit represents the complete random quantum circuit that is executed on the quantum computer. It includes all the gates and operations applied to the qubits in the experiment, while in elided circuit, some gates or operations may be omitted. and the patch circuit represents the full circuit is dividing it into two

equally independent subcircuits.

In the paper, Google claims that for 53 qubits 20 layers quantum circuit, sampling a million samples will take 200 seconds with average of 0.1% of fidelity, but meanwhile, for classical sampling will takes 10,000 years on a million cores to reach the same fidelity.

2.3.2 Gaussian Boson sampling

Boson sampling, a benchmark for quantum supremacy, demonstrates a quantum computer's ability to outperform classical computers. Proposed by Aaronson and Arkhipov in 2011 [21], it showcases linear-optical quantum computing's power by simulating non-interacting bosons in a network. The process involves input state preparation, a linear-optical network, photon detection, and sampling. Classical computers struggle with the problem's complexity, while quantum systems have a clear advantage. In 2020, USTC researchers claim to achieve quantum supremacy using Gaussian boson sampling with Jiuzhang [22]. Boson sampling highlights quantum computing's potential, stimulating further research despite limited practical applications.

This aspect of the work is primarily spearheaded by my colleague Zhenyan Zhao. In this dissertation, I will not delve into the details of Gaussian Boson sampling.

2.4 Fourier Analysis on RCS

In a quantum random circuit with n qubits, there are 2^n possible output states. The quantum random circuit is designed to generate a quantum state in which all qubits are entangled, the probability distribution of the output states may exhibit statistical properties similar to the Porter-Thomas distribution, and the average expectation of the square of the Fourier coefficients should be uniform under ideal circuit conditions. However, a direct correspondence on probability distribution should not be assumed, as the output distribution is influenced by various factors within the quantum random circuit.

However, this ideal uniform distribution is often not the case in actual quantum random circuits, as various factors, such as gate errors and decoherence, affect the final state distribution. The analysis of the output states in quantum random circuit sampling experiments is aimed at understanding the deviations from the ideal distribution and verifying the performance of the quantum computer.

Fourier analysis is a mathematical technique used to decompose a signal or function into its constituent frequencies, providing insights into the underlying structure and behavior of the data. In the context of RCS, Fourier analysis can be applied to the experimental results to study the properties of the probability distributions generated by the random circuits and reveal hidden patterns or correlations.

In my project, I conducted a comprehensive analysis of the results obtained from Google's quantum computing experiment by employing Fourier decomposition tech-

niques. I carefully examined the frequency components by categorizing the bit strings in the input space based on their Hamming weight. Details will be described in the next chapter.

Chapter 3

Boolean Function Analysis

In this chapter, I provided a detailed analysis of the Fourier coefficients of experimental results in quantum random circuit sampling on Google's experiment. I presented mathematical derivations and discussed the computation methods under different conditions, with or without noise. By examining the Fourier coefficients, we can gain a deeper understanding of the noise and fidelity situation in the quantum circuit.

3.1 Probability Distribution Function

For quantum random quantum circuits, the state of the output space could be represented by unitaries applying on the initial state. For any n-qubits quantum circuit, the final state $|\psi\rangle$ of the circuit could be represented by the following:

$$|\psi\rangle = \sum_{x\in\{0,1\}^n} A_x |x\rangle$$

and

$$\sum_{x\in\{0,1\}^n} |A_x|^2 = 1$$

Where x represent all possible output bitstring of the quantum circuit, A_x represent the amplitude of bitstring x. The probability of the circuit output bitstring x can be obtained by the following

$$p_\psi(x) = |\langle x|\psi\rangle|^2 = |A_x|^2$$

It can be found that the probability function is a **Real-valued Boolean Function**.

$$p(x) : \{0,1\}^n \to \mathbb{R}$$

3.2 Fourier Analysis

For every real-valued Boolean function, it can be expressed in terms of multi-linear polynomial function by applying **Walsh-Hadamard transform** or **Fourier expansion**

over the Boolean function. Therefore, probability function $p(x) : \{0,1\}^n \to \mathbb{R}$ could be rewrite in the following way.

$$p(x) : \{0,1\}^n \to \mathbb{R} = \sum_{s \in \{0,1\}^n} \hat{p}(s)\chi_s(x) \tag{3.1}$$

Where $\chi_s(x)$ is called **parity function**, given by the following.

$$\chi_s(x) = (-1)^{s \cdot x} \tag{3.2}$$

It is worth mentioning that in the above formula, s and x are bitstrings, and $s \cdot x$ represent the dot product of two bitstrings. Assume the length of s and x are n.

$$\begin{aligned} s = \{s_1 s_2 ... s_n\} \quad , \quad x = \{x_1 x_2 ... x_n\} \\ s \cdot x \quad = \quad \sum_k^n s_k \cdot x_k \end{aligned} \tag{3.3}$$

Through the definition, for real-valued Boolean function $p(x) : \{0,1\}^n \to \mathbb{R}$ and $q(x) : \{0,1\}^n \to \mathbb{R}$, the inner product could be written as the product of expectation of two functions.

$$\langle p(x), q(x) \rangle = \mathbb{E}\left[p(x)q(x)\right]$$

Furthermore, since $p(x)$ and $q(x)$ are real-valued Boolean function, the above formula could be rewrite into the following.

$$\langle p(x), q(x) \rangle = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} p(x)q(x) \tag{3.4}$$

Also for parity function $\chi_s(x)$, it can be fund that it is also a real-valued Boolean function, taking the input bitstring x and output a real number. Therefore, consider two different parity function $\chi_s(x)$ and $\chi_{s'}(x)$, by calculating their inner product,

$$\begin{aligned} \langle \chi_s(x), \chi_{s'}(x) \rangle &= \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \chi_s(x)\chi_{s'}(x) \\ &= \frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{s \cdot x}(-1)^{s' \cdot x} \\ &= \frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{(s+s') \cdot x} \\ &= \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{if } s \neq s' \end{cases} \\ &= \delta_{s,s'} \end{aligned}$$

Thus, parity function are orthogonal function.

And in the above mentioned formula, $\hat{p}(s)$ is the Fourier coefficient of s on function p. Since parity function are orthogonal, $\hat{p}(s)$ could be written as the following.

$$\begin{aligned}\hat{p}(s) &= \langle p(x), \chi_s(x) \rangle \\ &= \frac{1}{2^n} \sum_{x \in \{0,1\}^n} p(x)\chi_s(x) \end{aligned} \tag{3.5}$$

The Fourier coefficient $\hat{p}(s)$ shows the correlation of bitstring s with the sampling results. When noise is absent, the probability of receiving outcome s is $|\hat{p}(s)|^2$ [25]. But instead of using Fourier coefficient to represent the correlation between the probability function with the input space, to increase the precision, I will use **Correlator** given by the following formula, which indicate the re-scaled Fourier coefficient.

$$C(p(x);s) = 2^n \hat{p}(s) = \sum_{x \in \{0,1\}^n} p(x)\chi_s(x) \tag{3.6}$$

In the following section, I will show how to obtain the estimated Fourier coefficient on the data from Google's experiment on Random Circuit Sampling.

3.3 Analysis on RCS Experiment Data

Under ideal situation, where there is no noise in the circuit, the correlator of the output probability function of RCS when average over a uniform distribution, follow formula can be obtained.

$$\mathbb{E}[C(p;s)] = \frac{2^n - 1}{4^n - 1} \tag{3.7}$$

$$\text{For large } n : \mathbb{E}[C(p;s)] \approx \frac{1}{2^n} \tag{3.8}$$

Because of the fact that the sample size for each experiment is small, instead of look into every Fourier coefficients, I am combining the bitstring with same order k together and calculating its average. For every bitstring, define the **Order** of the bitstring s as $|s|$ which is the **Hamming weight** of the bitstring. In another words, it represent how many "1" contained inside the bitstring.

$$s = \{s_1 s_2 ... s_n\}, |s| = \sum_{k=1}^{n} s_k$$

It is obvious to see for order k, the **Fourier weights** of correlators is given by:

$$\begin{aligned} W_C^k[f] &= \sum_{s:|s|=k} C(p;s)^2 \\ &= \binom{n}{k} \frac{2^n - 1}{4^n - 1} \end{aligned} \tag{3.9}$$

Therefore, the expected value of Fourier weights under ideal circuit for order k is

$$\begin{aligned} \bar{W}_C^k[f] &= \frac{1}{\binom{n}{k}} \sum_{s:|s|=k} C^2(p;s) && (3.10) \\ &= \frac{2^n-1}{4^n-1} && (3.11) \end{aligned}$$

For every randomly generated circuits, Google repeated sampling for 100,000 times, and for every qubit size n, Google generated 10 different random circuits. But it is still difficult to obtain the true probability since there are 2^n numbers of possible output, which is much higher than 100,000 samples. I take the mean of the result over all samples as the average correlator of the bitstring. Suppose for every circuit, the sample size is m, therefore, the **experimental correlator** $\tilde{C}$ of bitstring s with the probability function on x could be written as the following.

$$\begin{aligned} \tilde{C}(p(x);s) &= \sum_{k=1}^{m} \frac{1}{m} \cdot \chi_s(x_k) \\ &= \frac{1}{m} \sum_{k=1}^{m} \chi_s(x_k) && (3.12) \end{aligned}$$

If sample size is infinite, the average correlator is equal to actual correlator.

$$\lim_{m\to\infty} \tilde{C}(p(x);s) = C(p(x);s) \tag{3.13}$$

In the actual quantum circuit, the circuit will infected by different sources of noise such as qubit decoherence, individual gate errors, and measurement errors, which leads to sampling error. The effect of this error on the correlator can be expressed mathematically as the following formula:

$$\tilde{C}(p;s) = e^{-\alpha|s|} C(p;s)$$

In the formula, α represent the error rate. It can be clearly found that this is a function of exponential decay, and the higher the α is, the smaller correlator value is. Therefore, in the analysis of the sampling results of the actual circuit, I expect an exponential decay image.

Combining formula 3.10 and 3.12, in one experiment, the Fourier weights $\bar{Z}(k)$ for order k can be given by the following formula.

$$\begin{aligned} \bar{Z}(k) &= \frac{1}{\binom{n}{k}} \sum_{s:|s|=k} \left(\tilde{C}^2(p(x_j);s)\right) \\ &= \frac{1}{m^2} \frac{1}{\binom{n}{k}} \sum_{s:|s|=k} \left(\sum_{j=1}^{m} \chi_s(x_j) \right)^2 && (3.14) \end{aligned}$$

At the same time, in order to observe the results expected by the above expectation formula of Fourier weight, combineing all random circuits samples and calculations together are necessary. Suppose the circuit number is D, the experimental Fourier weight $Z(k)$ can be given by the following.

$$Z(k) = \frac{1}{m^2}\frac{1}{D}\frac{1}{\binom{n}{k}}\sum_{i=1}^{D}\sum_{s:|s|=k}\left(\sum_{j=1}^{m}\chi_s(x_{ij})\right)^2 \tag{3.15}$$

m : Number of samples per experiment $= 100,000$

n : Qubit size $= 12, 14, 16, 18...$

D : Number of different experiments per qubit $= 10$

k : Order of bitstring $= 0$ to qubit number

Chapter 4

Code and Acceleration Method

In this chapter, I will describe the implementation process of the function described in chapter 3 on Python. I reduced the complexity of the functions using some special techniques and transferred them to the GPU, resulting in a 3800 times increase in speed. This achievement was significant, as it demonstrated the effectiveness of GPU computing in accelerating Fourier coefficient computation. By optimizing the code and utilizing the power of the GPU, I was able to achieve substantial performance improvements. Meanwhile, I will explain how I validate the correctness of my computation.

4.1 Programming Language

Python is a high-level programming language that is widely used for web development, scientific computing, data analysis, artificial intelligence, and many other applications. It is particularly useful for data analysing is because of its simplicity, readability, and availability of various libraries and tools. In addition, Python is popular for GPU computing because of its rich ecosystem for GPU computing, including CuPy I am using in this dissertation. Meanwhile, Python is also adept at data visualization. In this project I will use Matplotlib to display the result of the experiment.

4.1.1 Matplotlib

Matplotlib is a powerful library used to display all kinds of diagrams and chart. For visualizing data, Matplotlib has the following advantages:

Flexibility: Matplotlib provides a wide range of plots options, also allow user to customize everything with simple procedure, including color, font, line style, label, etc. Integrative.

Integrative: Matplotlib can be perfectly combined with most of the data analyse library and data structure, like CuPy array or Pandas dataframe.

Huge user base: Matplotlib the most popular visualization library in the world, with

huge user base, it is very convenient to find relevant knowledge and tutorials.

4.1.2 CuPy

CuPy is a GPU-accelerated library for numerical computations, designed to be compatible with NumPy in gramma. For CuPy, it has following advantages:

Speed: CuPy call the computing power of GPU, which could significantly increase the speed of calculation comparing with CPU based library like NumPy.

Usability: CuPy has exactlly same interface with NumPy, which greatly reduce the cost of learning. This makes it very simple for those familiar with NumPy to adopt CuPy for GPU acceleration.

Flexibility: Comparing with other GPU based library like PyTorch or TensorFlow, CuPy is more lightweight which only focus on numerical computations, while other two are more focusing on deep learning. Also, CuPy could be easily combined with other Python libraries like Matplotlib.

4.2 Calculation Process

In this project, I conduct a variety of experiments and analyze the results from multiple perspectives. The core component of the code is the implementation of the aforementioned formula.

$$Z(k) = \frac{1}{m^2}\frac{1}{D}\frac{1}{\binom{n}{k}}\sum_{i=1}^{D}\sum_{s:|s|=k}\left(\sum_{j=1}^{m}\chi_s(x_{ij})\right)^2$$

In practice the formula can be broken down into the following steps:

1. Load a single experimental dataset into memory.
2. Assuming a qubit size of n, generate an empty list of length $n+1$ to represent the results of different orders.
3. Iterate over 2^n bitstrings. For each bitstring, if its order is k, compute the square of the sum of the parity function outcome with every bitstring from the previously loaded experimental data. Add the value to the k^{th} element of the list.
4. Going to the next experiment, repeat from step 1 to step 3 until finishing all experiments, then multiply each element in the list by $\frac{1}{m^2}\frac{1}{D}\frac{1}{\binom{n}{k}}$.
5. Plot the diagram and save the result into a text file.

4.3 Complexity Analysis

It can be found that the time complexity of the formula can be express as following.

$$\chi_s(x_{ij}) \quad : \quad O(n) \tag{4.1}$$

$$\left(\sum_{j=1}^{m} \chi_s(x_{ij})\right)^2 \quad : \quad O(m \cdot n) \tag{4.2}$$

$$\sum_{s:|s|=k} \left(\sum_{j=1}^{m} \chi_s(x_{ij})\right)^2 \quad : \quad O(m \cdot n \cdot 2^n) \tag{4.3}$$

$$\sum_{i=1}^{D} \sum_{s:|s|=k} \left(\sum_{j=1}^{m} \chi_s(x_{ij})\right)^2 \quad : \quad O(m \cdot n \cdot 2^n \cdot D) \tag{4.4}$$

$$Z(k) \quad : \quad O(m \cdot n \cdot 2^n \cdot D) \tag{4.5}$$

4.1 Parity function, with complexity of $O(n)$ cause it will need to calculate every bit in bitstring with length n. Details described in section 4.4.1.

4.2 The complexity of process one bitstring with every samples in the experiment.

4.3 This step involves calculating all bitstrings with order k. Although the complexity of this step should be dependent on k, ultimately, all 2^n bitstrings will be processed. Therefore, from a macro perspective, the increased complexity of this step is 2^n.

4.4 Going over D different experiment circuit will cause the increase of complexity of D.

4.5 Final overall complexity.

Therefore, the time complexity of the function is $O(m \cdot n \cdot 2^n \cdot D)$. In the next section, I will describe a way to optimize the time complexity and method to speedup with GPU.

4.4 Speedup Method

In this section, I will delve into the details of the code implementation, describing the techniques employed to reduce time complexity and actual runtime.

4.4.1 Parity Function

This is a straightforward implementation of the parity function, by going over all bit from the bitstring and apply the multiplication then add together.
The parity function is implemented in a straightforward manner by iterating over all bits of the bitstring, applying the multiplication operation, and summing the results. The time complexity of this function is $O(n)$, where n represents the bitstring length or qubit size. By employing the following method, the time complexity can be reduced to $O(1)$.

Algorithm 1 A Naive Parity Function Implementation (bitstring1, bitstring2)

Require: $|\text{bitstring1}| = |\text{bitstring2}|$ ▷ represent s and x in formula 3.2
  $result \leftarrow 0$
  $index \leftarrow 0$
  while $index < |bitstring1|$ **do**
    $bit1 \leftarrow bitstring1[index]$ ▷ represent the bit at the *index* position in bitstring1
    $bit2 \leftarrow bitstring2[index]$ ▷ represent the bit at the *index* position in bitstring2
    $result \leftarrow result + bit1 \cdot bit2$
    $index \leftarrow index + 1$
  end while
  return -1^{result}

The calculation of the parity function is divided into three parts, which are dot product, bit sum and -1 exponent. I will describe their optimization methods separately.

4.4.1.1 Dot Product on Bitstring

For bitstring $s = \{s_1 s_2 ... s_n\}, x = \{x_1 x_2 ... x_n\}$, the dot product is $s \cdot x = \sum_k^n s_k \cdot x_k$ as described in formula 3.3. It is obvious that for two bit s_n and x_b, the result of $s_n \cdot x_n$ is equivalent with s_n AND x_n. On classical computer architecture, the time complexity of bitwise AND is $O(1)$. Therefore, for the first step of calculating the dot product of two bitstrings, I could just simply apply AND operations on two bitstrings. By first converting the bitstring into binary digits, then applying AND operation between two binary digits.

11	00	10	11	00
&&	&&	&&	&&	&&
10	01	01	10	11
10	00	00	10	00

4.4.1.2 Summing Bits and Exponential Operation

Then, the process of summing the bits can be viewed as determining the Hamming Weight of the bitstring 1000001000 in the given example, in another word, the order. Because in the whole calculation process, the order of 2^n bit strings needs to be calculated anyway. If these calculation results are stored, there is no need to repeat the calculation process. Consequently, by pre-calculating the order of all possible bitstrings, the summing process can be simplified to accessing the result from the bitstring-order list and then computing the exponent of -1.

By incorporating the aforementioned techniques, a more efficient method for computing the parity function has been developed, as shown below. Although this function requires pre-computation of the order list for 2^n bitstrings, it reduces the time complexity of parity to $O(1)$. Thus, the total time complexity of the overall computation is reduced from $O(m \cdot 2^n \cdot D \cdot n)$ to $O(m \cdot 2^n \cdot D + 2^n)$, which is equivalent to $O(m \cdot 2^n \cdot D)$.

Algorithm 2 A Fast Way to Compute Parity Function

▷ This function **int_to_bin_list** used to transfer a integer to its binary form, and stored in a list. Eg.: input 9 will output [1,0,0,1]. The purpose of this is to faster calculate the Hamming weight of the bitstring in the following step.

procedure INT_TO_BIN_LIST(A: int)
  $binaryA \leftarrow$ INT_TO_BINARY(A)
  $listBinaryA \leftarrow$ SPLIT($binaryA$)
  return $listBinaryA$
end procedure

$bin\_order \leftarrow [\,]$ ▷ List store the order for all bitstring
$A \leftarrow 0$
while $A \leq 2^n$ **do**
  $listBinaryA \leftarrow$ INT_TO_BIN_LIST(A)
  $orderA \leftarrow$ SUM($listBinaryA$) ▷ Calculate the order the bitstring
  $bin\_order \leftarrow bin\_order + [orderA]$ ▷ Append the result into result list
end while

procedure PARITY_FUNCTION(bitstring1, bitstring2)
  $int1 \leftarrow$BINARY_TO_INT($bitstring1$) ▷ Transfer the bitstring into integer
  $int2 \leftarrow$BINARY_TO_INT($bitstring2$) ▷ Transfer the bitstring into integer
  $sum \leftarrow bin\_order[int1 \;\&\; int2]$ ▷ The Dot Product of two bitstring
  $result \leftarrow -1^{sum}$ ▷ Parity function result
  return $result$
end procedure

4.4.2 GPU Approach

GPUs are designed to handle multiple tasks simultaneously by allowing the faster execution of parallelizable calculations compared to CPUs. In this project, the computation could be converted to parallel computing on CuPy array instead of original loop structure by slightly modify the storage structure.

Like the original structure, the code still pre-calculate the *bin_order* list, but instead of the original list structure, CuPy arrays are used to store the list data.

```
bin_order = cupy.array
([
    int(sum(int_to_bin_list(i))) for i in range(0, sLength)
])
```

Additionally, another array called *sBarList* is introduced to store bitstrings with the same order together. The length of *sBarList* is $n+1$, with the element at index 0 storing

all bitstrings of order 0, the element at index 1 storing all bitstrings of order 1, and so on.

```
sBarList = [[] for i in range(0, n + 1)]

for i in range(0, 2**n):
    sBarList[sum(int_to_bin_list(i))].append(i)
sBarList = [cupy.array(i) for i in sBarList]
```

By arranging the array in this manner, instead of iterating from $00..00(0)$ to $11..11(2^n)$, even though the order of all bitstrings has been pre-calculated, running on the GPU causes the communication delay between memory and GPU to become the bottleneck of the calculation. Although at the end of every parity function calculation there will still be a memory access to check the order of the summing result, this method reduces the overall runtime. As a result, the calculation function for a single experiment sample result can be written as follows.

```
def calculate(filename, n, m, bin_order, sBarList):
    Z_file = [0 for i in range(n + 1)]
    f = open(filename, "r")
    fileListLines = f.readlines()
    fileList = cp.array([bin_to_int(i) for i in fileListLines])
    indx = 0
    for sBar_num in range(len(sBarList)):
        sBar = sBarList[sBar_num]
        for s_num in range(len(sBar):
            s = sBar[s_num]
            temp = cp.bitwise_and(s, fileList)
            bin0 = lambda x: bin_order[x]
            temp = (-1)**(bin0(temp))
            Z_file[indx] = Z_file[indx] + int(cp.sum(temp))**2
        indx = indx + 1
    return (Z_file)
```

The core component for achieving acceleration is this part:

```
            temp = cp.bitwise_and(s, fileList)
            bin0 = lambda x: bin_order[x]
            temp = (-1)**(bin0(temp))
```

It can be proved by experiments on patch circuit that this method improves the operation speed by 3828 times.
(Both experiments were running on the same machine. Configured as Intel i9-11900k, 32G memory, NVIDIA RTX 3090)

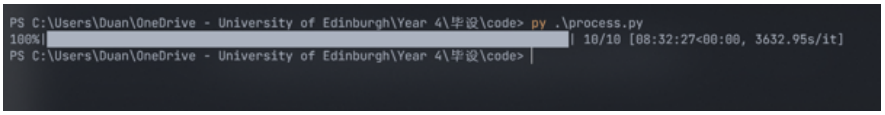


Figure 4.1: The original CPU-based design took 8 hours and 32 minutes to process the 12 qubits of data for all 10 experiments.

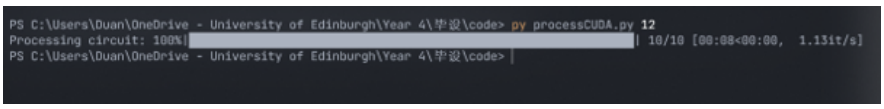


Figure 4.2: The improved GPU-accelerated design takes only 8 seconds to process the same data.

4.4.3 Performance Evaluation

The GPU-accelerated code's runtime is summarized in the table for 12 to 24 qubits on patch circuit. In general, for every 2 additional qubits, the required runtime increases by a factor of 3.5. For 24 qubits, it takes about five hours.

Qubit size	Running time
12	8 sec
14	26 sec
16	120 sec
18	476 sec
20	1537 sec
22	5391 sec
24	18920 sec

Table 4.1: Running Time Table

By analogy, 26 qubits will take about 17 hours, and 28 qubits will take about 60 hours. Simultaneously, it was observed that during the entire operation process, video memory usage remained low, occupying less than 2Gb of video memory. This offers the potential for optimization. When calling the GPU to perform calculations, the communication time with the memory and the CPU limits the calculation speed. If more data can be stored in video memory, the speed and time of the calculation may be further optimized. The next section will describe some possible ways to achieve further optimization.

4.5 Further Potential Improvement

4.5.1 Broadcasting Technique

The NVIDIA RTX 3090 has 24Gb of V-Ram, which is relatively small. Thus, the code iterates over each bitstring one by one and performs parallel computing with all experimental data simultaneously, as shown in the following code:

```
    for sBar_num in range(len(sBarList)):
        sBar = sBarList[sBar_num]
        for s_num in range(len(sBar)):
            s = sBar[s_num]
```

However, with a computing resource offering higher V-Ram, it is possible to treat all bitstrings with the same order (written as b_k) as an array and calculate the matrix of the **outer and** (written as **&&** below) with the experimental data array (written as e_m).

$$\begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_k \end{bmatrix} \&\& \begin{bmatrix} e_1 & e_2 & \dots & e_m \end{bmatrix} = \begin{bmatrix} b_1 \& e_1 & b_1 \& e_2 & \dots & b_1 \& e_m \\ b_2 \& e_2 & b_1 \& e_2 & \dots & b_2 \& e_m \\ \dots & \dots & \dots & \dots \\ b_k \& e_k & b_1 \& e_2 & \dots & b_k \& e_m \end{bmatrix}$$

To achieve this with CuPy, broadcasting techniques can be used. Broadcasting techniques in NumPy and CuPy enable efficient arithmetic operations on arrays of different shapes and sizes. In this project, this technique could potentially be used to achieve further speedup by avoiding memory access when iterating over bitstrings.

4.5.2 Avoid Redundant Computation

When examining the experimental data more closely, it is evident that many redundant bitstrings exist in the results. For each qubit size, there are 1,000,000 total samples (for patch circuit). The unique bitstring numbers for 12 to 38 qubits are summarized in the following table.

Qubit size	Unique Bitstring Number on Experiment
12	4096
14	16320
16	57845
18	128716
20	176707
22	194021
24	198471
26	199617
28	199914
30	199979
32	199997
34	199998
36	200000
38	199999

Table 4.2: Unique Bitstring Number

The table reveals that, compared to the total sample number (1,000,000), a significant number of redundant bitstrings exist in the experimental data, meaning that more than $4/5$ of the calculations are redundant. To avoid this, the calculation result could potentially be saved in the V-Ram, and each calculation could be checked to determine whether it had been calculated previously. In theory, this could save a considerable amount of time. However, the actual performance can only be determined through experimentation.

4.6 Validation

4.6.1 Toy Model

To verify the correctness of the code, a Toy Model constructed by my colleague Sirui Chen was used, and I applied my calculation on it to verify it the result matches with the expectation of the Toy Model. The purpose of the toy model is to simulate the noise of a classical stochastic process. The toy model takes $\bar{0}$ bitstring as input and, for every bit in the bitstring, applies a random error effect. Specifically, the model takes an error possibility parameter p $(0 \leq p \leq 1)$. For each bit, an error occurs with probability p, and when the error occurs, the bit will change into 1 or 0 with equal probability.

4.6.2 Validation on Toy Model

Applying the previously mentioned calculations to the generated samples from the toy model, the average correlators' weights should follow the formula:

$$\bar{W}_c^k = (1-p)^{2k}$$

I applied the results with a sample size of $1,000,000$ for $p = 0.5/0.6/0.7/0.8$ into my calculation model and compared them with the expected results. The following graph was obtained.

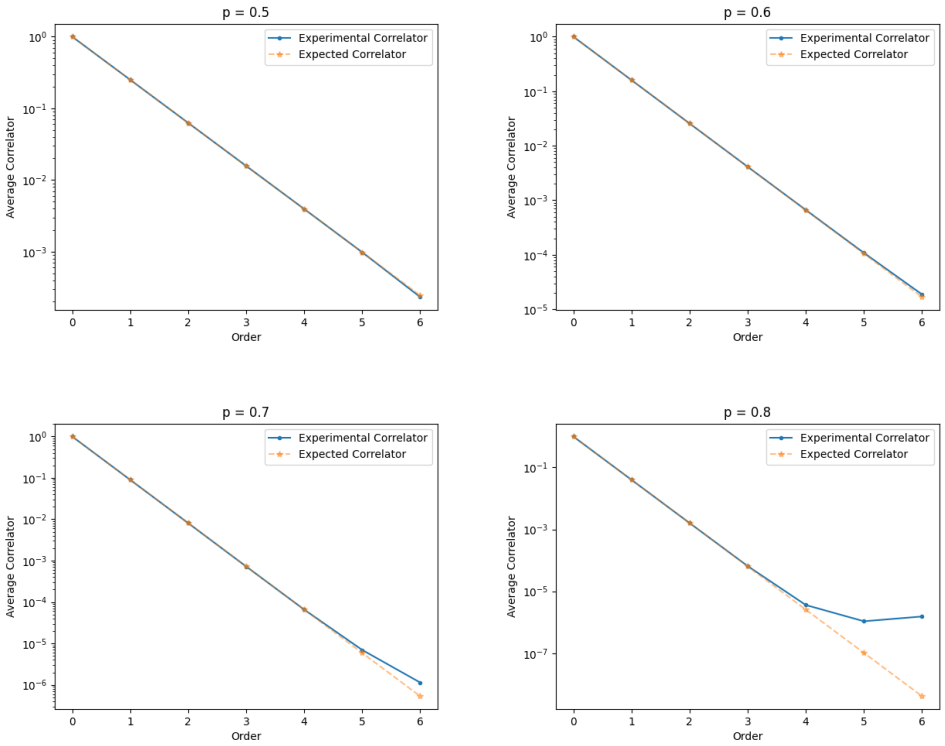


Figure 4.3: Toy Model Experimental Average Correlator Weight and Expectation Correlator

The graph shows that most of the experimental average weights fit the expectation well,

proving the correctness of the calculation model. However, for $p = 0.8$, the higher-order experimental correlators deviate from the expected value, which is likely due to the limited sample size. As mentioned earlier, only when an infinite number of samples are used, the experimental correlators equal the actual correlators.

$$\lim_{m\to\infty} \tilde{C}(p(x);s) = C(p(x);s)$$

I also compared different sample sizes of 1000, 10000, 100000, and 1000000 and confirmed this conjecture. The details will be described in the next chapter.

4.7 Conclusion

In conclusion, the improvements to the code and the introduction of GPU acceleration have significantly reduced the time complexity and runtime of the calculations. Additionally, the use of CuPy arrays and broadcasting techniques, as well as optimization for higher V-Ram computing resources, have further enhanced the code's efficiency. Finally, the verification of the calculation model using a Toy Model has demonstrated the correctness of the implemented improvements. In the next chapter, I will present the results obtained using the improved code, along with an in-depth data analysis and error analysis to further understand the implications of noise in the quantum circuit.

Chapter 5

Experiments and Results

5.1 Overview and Hypothesis

I carried out an analysis of Google's random circuit sampling results obtained from the Sycamore quantum processor. My primary focus was on examining both the Patch circuit and Full circuit configurations independently, utilizing my calculation model to process the data. I evaluated the outcomes for circuits ranging from 12 to 24 qubits in size and performed a comparative analysis of the results to gain a deeper understanding of their implications. Upon completing the analysis of Google's random circuit sampling results from the Sycamore quantum processor, I observed the performance trends and potential correlations between the Patch circuit and Full circuit configurations. By scrutinizing these configurations across varying qubit sizes, from 12 to 24 qubits, I sought to uncover the impact of qubit count on the accuracy of the quantum computations. This detailed investigation provided valuable insights into the strengths and limitations of the Sycamore processor, as well as informed the ongoing discussion surrounding quantum advantage and the broader field of quantum computing.

As discussed in Section 3.3, the distribution of RCS circuits is expected to exhibit exponential decay, which also serves as the hypothesis for this experiment. Concurrently, Section 4.6.2 highlights that the sample size and the number of qubits significantly influence the resulting image. Based on the insights from Section 4.6.2, it can be anticipated that larger qubits will require more samples to approximate the actual result accurately. In other words, for higher qubit numbers, the current sample size may be insufficient to accurately represent the real correlator outcomes.

5.2 Data Collection and Analysis

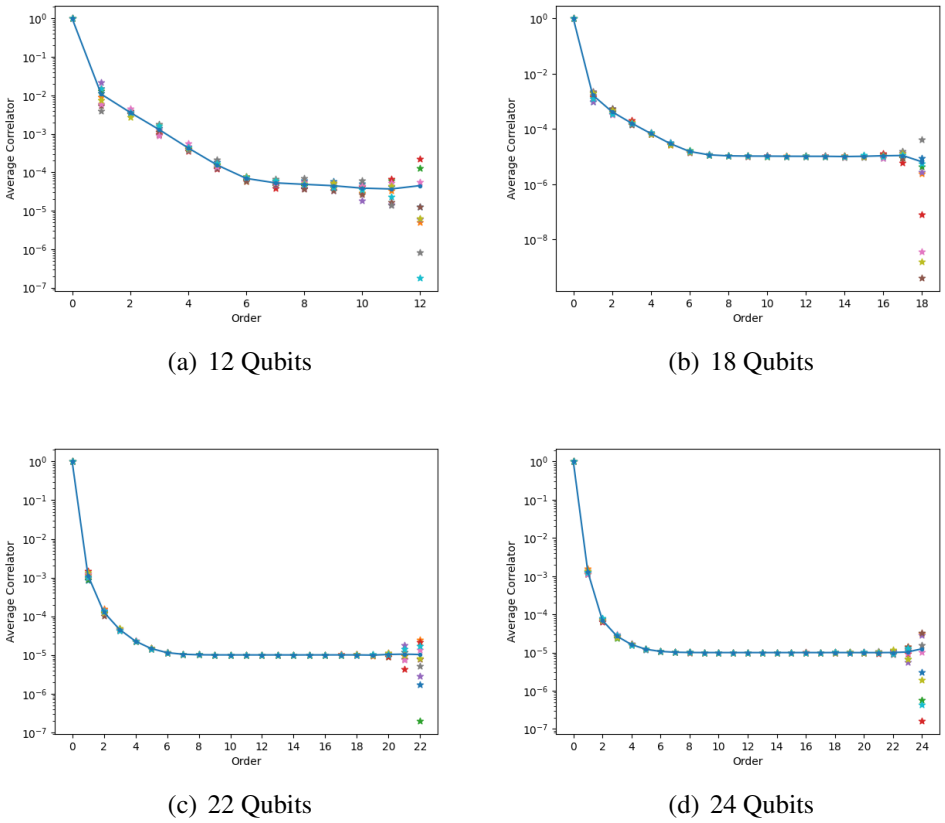
5.2.1 Data Collection and Description

The experimental dataset was acquired from DRYAD (`https://datadryad.org/stash/dataset/doi:10.5061/dryad.k6t1rj8`), encompassing data for quantum circuits ranging from 12 to 53 qubits in size. Each qubit size featured 10 randomly

generated quantum circuits for Full circuit, Elided circuit, and Patch circuit configurations, represented in Python, QISKit, and QASM formats. Additionally, every circuit included a text file containing the sampled bitstring results. The number of samples varied depending on the circuit type. In this analysis, I focused on the Full circuit and Patch circuit configurations, which contained 500,000 and 100,000 samples per circuit, respectively.

5.2.2 Patch Circuit Result

The results of the Fourier analysis for 12, 18, 22 and 24 qubits are shown below.



(a) 12 Qubits (b) 18 Qubits

(c) 22 Qubits (d) 24 Qubits

Figure 5.1: Experimental results for 12, 18, 22 and 24 qubits are shown. Dots represent results in a single circuit, lines represent the average of all circuits with the same qubits

Analysis of the image distributions reveals that they are highly similar, exhibiting an exponential decay on a logarithmic scale. Notably, for low order correlator weights, the experimental results are very similar, whereas for higher order weights, some variation appears to be present. Nevertheless, after averaging the results of all experiments, a smooth trend remains evident in the curve.

In order to investigate the relationship between the trend and qubit size, I created a graph by plotting the average result of each qubit. The resulting graph is shown below. A clear trend can be seen from the graph, with larger qubit sizes exhibiting smaller

average correlator weights. This shows that for circuits with more qubits, the resulting output is less correlated on the overall qubit state, in the other word, noise increased.

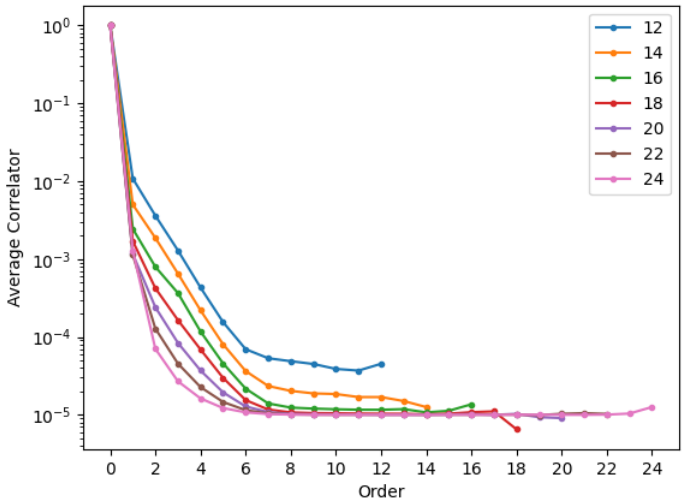
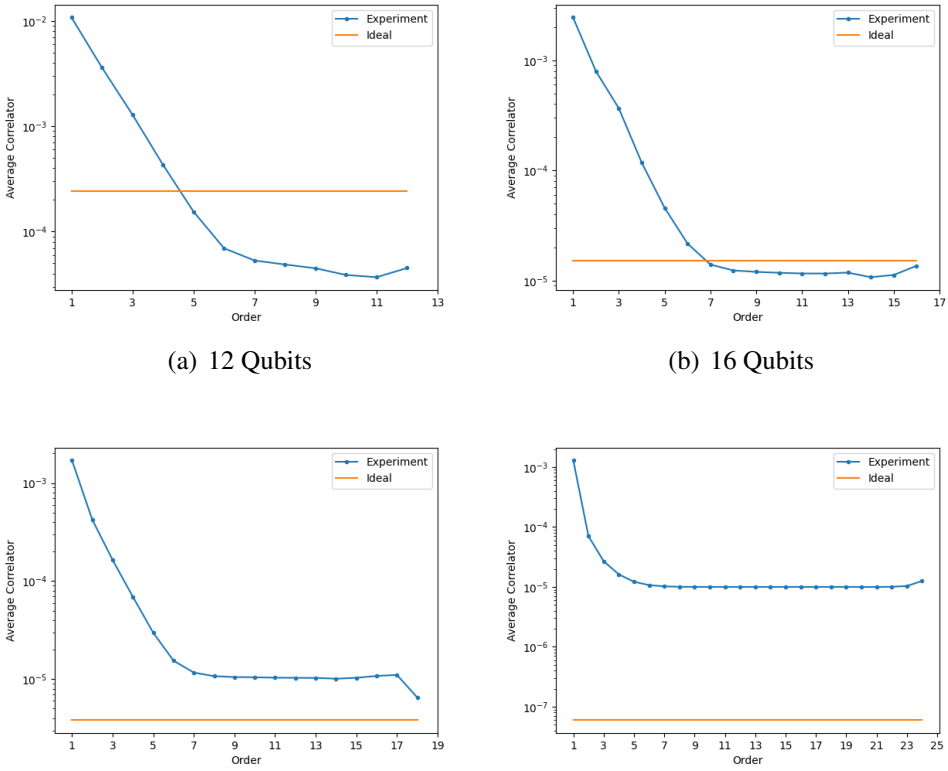


Figure 5.2: Average Correlator Weight on 12-24 Qubits

Also, I plotted the expected curves for each qubit size under ideal conditions and compared them with the previous graph. According to the formula in Chapter 3, in an ideal case, the average correlator weight in RCS for each order should be equal to $1/2^n$, except for order 0, which equals 1. In order to look clearer, I omitted the part where the order is 0, since they are all equal to 1.



(a) 12 Qubits (b) 16 Qubits

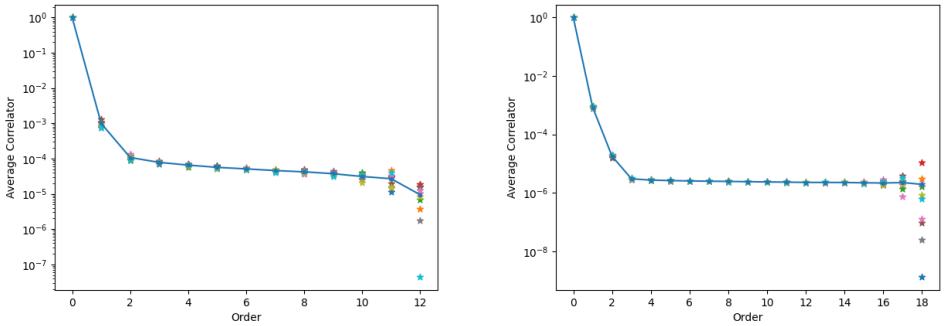
(c) 18 Qubits (d) 24 Qubits

Figure 5.3: Experimental results for 12-24 qubits are shown, with the ideal situation result

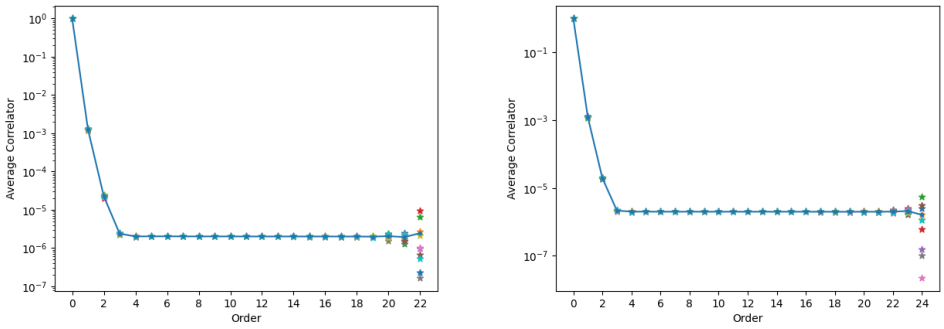
Surprisingly, the results demonstrate that for circuits with a larger number of qubits, the correlator weights are, in fact, higher than those in the ideal noise-free case. This observation indicates that the degree of correlation in the experimental correlator exceeds that of the noise-free circuit. Conversely, for circuits with fewer qubits, experimental results are lower than ideal values, with the exception of smaller number orders. This suggests that in circuits with a smaller number of qubits, lower-order correlators exhibit deviations from the ideal circuit behavior beyond the hypothesized exponential decay with order. In circuits with a higher number of qubits, lower-order correlators display similar effects, but the results for high-order correlators remain inconclusive. This counterintuitive finding warrants further investigation to uncover the underlying reasons for this discrepancy. I also made a conjecture, which will be elaborated in section 5.3.

5.2.3 Full Circuit Result

For the full circuit, I drew the same diagram. I found that the overall trend is very similar to patch circuit.



(a) 12 Qubits (b) 18 Qubits



(c) 22 Qubits (d) 24 Qubits

Figure 5.4: Experimental results for 12-24 qubits are shown. Dots represent results in a single circuit, lines represent the average of all circuits with the same qubits

Comparing the data of different qubits together, I can also get the same conclusion as the patch circuit, the larger the qubit, the more noise.

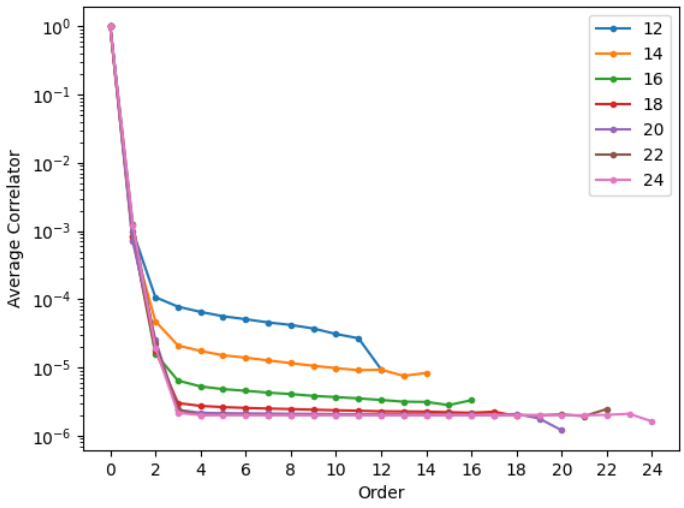
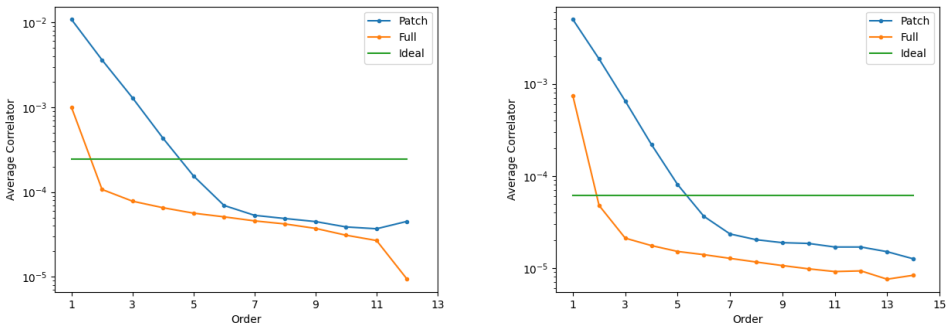


Figure 5.5: Average Correlator Weight on 12-24 Qubits

I combined the results of both patch and full circuits in a single graph to observe the trends more clearly. Overall, the values for the patch circuit are higher than those for the full circuit, and full circuit decays faster than patch circuit in terms of k. However, the full circuit also exhibits a similar issue: as the qubit size increases, the results surpass the ideal situation. Yet, unlike the patch circuit, the full circuit's deviation from the ideal situation occurs more gradually. In the full circuit results, it is evident that the outcome for 18 qubits remains below the ideal value, while in the patch circuit, the result for 18 qubits already exceeds the ideal expectation. At the same time, it can also be found that for the full circuit, except those cases where the correlator results are higher than ideal due to sample size error, other values are lower than ideal, except order 1, which is also fit with the previous expectation, because the hypothesis of $1/2^n$ is only proven theoretically for full circuits.



(a) 12 Qubits (b) 14 Qubits

(c) 18 Qubits (d) 24 Qubits

Figure 5.6: Experimental results for 12-24 qubits are shown, with the ideal situation result

5.3 Error Analysis

In the hypothesis, I mentioned that the number of samples will have a certain impact on the data of larger qubits. To elucidate this peculiar phenomenon and verify this hypothesis, I conducted additional experiments and discovered that the number of experimental samples plays a crucial role in the occurrence of this anomaly. Initially, I

focused on the patch circuit. I randomly divided the circuit results into several subsets, with the number of subsets ranging from 5 to 10, and treated each group as a separate experimental outcome for analysis.

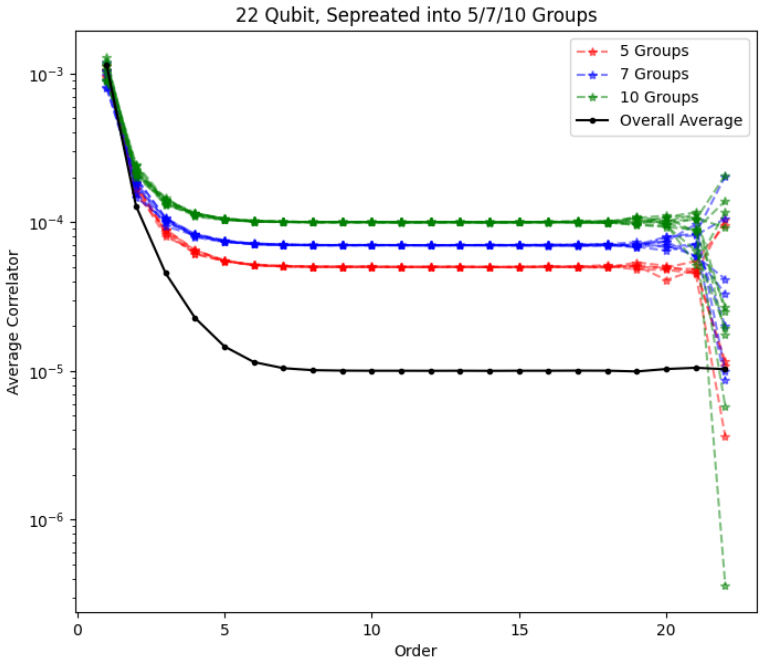


Figure 5.7: Comparison of Experimental Outcomes

For the classifications of 5, 7, and 10, the number of samples in each group are 20,000, 14,285, and 10,000, respectively, arranged in descending order. As illustrated in the image, it is evident that the 10-class group with the fewest samples per class exhibits the highest weight, followed by the 7-class and 5-class groups. This further demonstrates that a larger number of samples brings the weight closer to the true value, resulting in a smaller weight. This is also the reason why the overall value of the Full circuit is lower than that of the Patch circuit since the Full circuit contains 500,000 samples per experiment while 100,000 in Patch circuit. Simultaneously, in conjunction with previous conclusions, it can be inferred that a smaller qubit size requires fewer samples. As depicted in Figure 5.6, the results for smaller qubit size appear more reasonable.

This conclusion can also provide insight into the phenomenon discussed in Section 4.5.2 to some extent. By incorporating the P=0.8 situation depicted in Figure 4.3 into various sample sizes for illustration, the resulting image can be observed below.

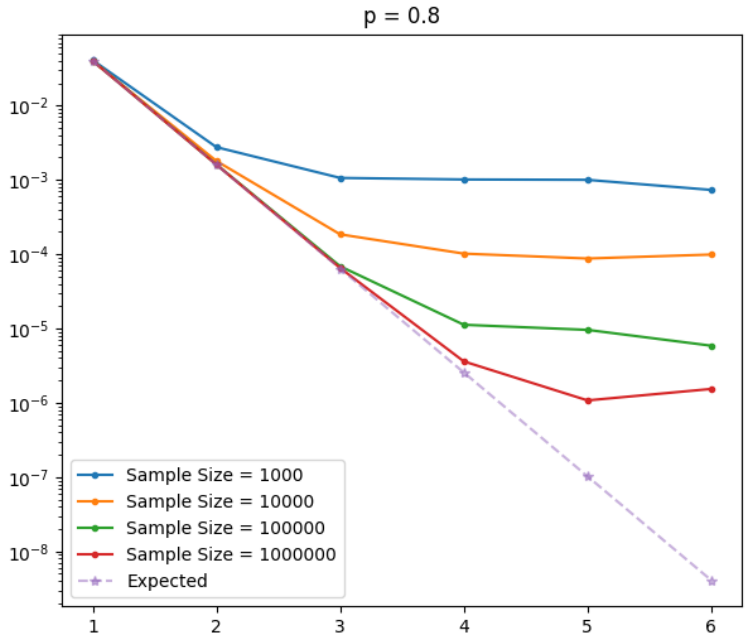


Figure 5.8: Comparison of Different Sample Size on Toy Model

It can be observed that as the sample size increases, the result approaches the expected performance more closely. Additionally, it is worth noting that under the same sample size, the group with a larger error exhibits a more significant deviation from the expectation. This suggests that for greater noise levels, a larger sample size may be required to approach the true value more accurately.

Chapter 6

Summary and Conclusion

In this dissertation, my primary focus is on the in-depth analysis of Google's quantum random circuit sampling experiment, which plays a pivotal role in evaluating the company's claims surrounding quantum supremacy. To comprehensively assess these assertions, I employ the method of Fourier analysis, a powerful tool that allows for a thorough investigation of the relationship between the experimental results and the number of qubits involved. By doing so, I aim to ascertain the robustness of Google's claims and better understand the implications of their findings in the context of quantum supremacy.

Through experimentation, I obtained the expected results for circuits with smaller qubit sizes, which demonstrated an exponential decay in the images. Observing both the full circuit and the patch circuit independently, they both conform to the anticipated exponential decay, with the full circuit exhibiting a more rapid decay rate. Additionally, I observed that for small orders in small qubit size circuit, the correlators have higher values than ideal line $1/2^n$. This unexpected finding presents an intriguing point for future research. The discrepancy may be attributed to the assumption of $1/2^n$ might only true for a large number of qubits, or it could be due to noise mechanisms beyond those responsible for producing the exponential decay of correlators. At the same time, I stumbled upon a peculiar and unexpected phenomenon. My experimental results demonstrated that the correlator values obtained from Google's experiments with high qubit size were, in fact, higher than those derived from a theoretically noise-free ideal circuit and approaches to a horizontal line. This counterintuitive observation prompted me to delve deeper into the data, conducting a rigorous analysis to uncover the root cause behind this anomaly. After extensive examination, it became apparent that the primary factor contributing to this discrepancy was an insufficient number of samples utilized in the experiments. Furthermore, through the Toy model, I discovered that the degree of noise present within the system played a significant role in influencing the outcomes related to different sample sizes. These findings indicated that an interplay between noise levels and sample sizes had a direct impact on the experimental results.

Given these insights, a crucial research question has emerged: How many sample sizes are needed to obtain correlator values that closely resemble their true values in a quantum random circuit sampling experiment? And clearly through my work, the

sample size for Google is not enough. Addressing this question is of utmost importance for refining our understanding of quantum supremacy and improving the accuracy and reliability of future experiments. By exploring this avenue of research, we can contribute valuable knowledge to the ongoing discussion surrounding quantum advantage and ultimately advance the broader field of quantum computing.

Bibliography

[1] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[2] Chelsea Whyte. What next for quantum computers? *New Scientist*, 243(3250):15, 2019.

[3] Richard P Feynman. Simulating physics with computers. In *Feynman and computation*, pages 133–153. CRC Press, 2018.

[4] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[5] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992.

[6] Daniel R Simon. On the power of quantum computation. *SIAM journal on computing*, 26(5):1474–1483, 1997.

[7] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[8] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[9] Isaac L Chuang, Neil Gershenfeld, and Mark Kubinec. Experimental implementation of fast quantum searching. *Physical review letters*, 80(15):3408, 1998.

[10] Yasunobu Nakamura, Yu A Pashkin, and JS Tsai. Coherent control of macroscopic quantum states in a single-cooper-pair box. *nature*, 398(6730):786–788, 1999.

[11] Lieven MK Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S Yannoni, Mark H Sherwood, and Isaac L Chuang. Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414(6866):883–887, 2001.

[12] Stephan Gulde, Mark Riebe, Gavin Lancaster, Christoph Becher, Jürgen Eschner, Hartmut Häffner, Ferdinand Schmidt-Kaler, Isaac L Chuang, and Rainer Blatt. Implementation of the deutsch–jozsa algorithm on an ion-trap quantum computer. *Nature*, 421(6918):48–50, 2003.

[13] C Negrevergne, TS Mahesh, CA Ryan, M Ditty, F Cyr-Racine, W Power, N Boulant, T Havel, DG Cory, and R Laflamme. Benchmarking quantum control methods on a 12-qubit system. *Physical review letters*, 96(17):170501, 2006.

[14] JH Plantenberg, PC De Groot, CJPM Harmans, and JE Mooij. Demonstration of controlled-not quantum gates on a pair of superconducting quantum bits. *Nature*, 447(7146):836–839, 2007.

[15] Alberto Politi, Jonathan CF Matthews, and Jeremy L O'Brien. Shor's quantum factoring algorithm on a photonic chip. *Science*, 325(5945):1221–1221, 2009.

[16] Matteo Mariantoni, Haiyan Wang, Takashi Yamamoto, Matthew Neeley, Radoslaw C Bialczak, Yu Chen, Mike Lenander, Erik Lucero, Aaron D O'Connell, Daniel Sank, et al. Implementing the quantum von neumann architecture with superconducting circuits. *Science*, 334(6052):61–65, 2011.

[17] Thomas Monz, Daniel Nigg, Esteban A Martinez, Matthias F Brandl, Philipp Schindler, Richard Rines, Shannon X Wang, Isaac L Chuang, and Rainer Blatt. Realization of a scalable shor algorithm. *Science*, 351(6277):1068–1070, 2016.

[18] John Preskill. Quantum computing and the entanglement frontier. *arXiv preprint arXiv:1203.5813*, 2012.

[19] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J Bremner, John M Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *Nature Physics*, 14(6):595–600, 2018.

[20] Adam Bouland, Bill Fefferman, Chinmay Nirkhe, and Umesh Vazirani. On the complexity and verification of quantum random circuit sampling. *Nature Physics*, 15(2):159–163, 2019.

[21] Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 333–342, 2011.

[22] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, et al. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020.

[23] Lars S Madsen, Fabian Laudenbach, Mohsen Falamarzi Askarani, Fabien Rortais, Trevor Vincent, Jacob FF Bulmer, Filippo M Miatto, Leonhard Neuhaus, Lukas G Helt, Matthew J Collins, et al. Quantum computational advantage with a programmable photonic processor. *Nature*, 606(7912):75–81, 2022.

[24] James King, Sheir Yarkoni, Jack Raymond, Isil Ozfidan, Andrew D King, Mayssam Mohammadi Nevisi, Jeremy P Hilton, and Catherine C McGeoch.

Quantum annealing amid local ruggedness and global frustration. *Journal of the Physical Society of Japan*, 88(6):061007, 2019.

[25] Michael J Bremner, Ashley Montanaro, and Dan J Shepherd. Achieving quantum supremacy with sparse and noisy commuting quantum computations. *Quantum*, 1:8, 2017.