Exploring the Limits of Classical Simulation: From Computational Many-Body Dynamics to Quantum Advantage

by

Gregory Donald Kahanamoku-Meyer

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Physics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Norman Y. Yao, Chair
Professor Umesh V. Vazirani
Professor Joel E. Moore

Summer 2023

Exploring the Limits of Classical Simulation:
From Computational Many-Body Dynamics to Quantum Advantage

Abstract

Exploring the Limits of Classical Simulation:
From Computational Many-Body Dynamics to Quantum Advantage

by

Gregory Donald Kahanamoku-Meyer

Doctor of Philosophy in Physics

University of California, Berkeley

Professor Norman Y. Yao, Chair

For many years after the dawn of computing machines, it seemed to be the case that the dynamics of any physical system (including all computers, which of course are physical systems themselves) could be efficiently simulated by a simple model of a computer called the Turing machine. A consequence was that computational problems that were found to be hard for Turing machines—those requiring a runtime superpolynomial in the size of the input—remained hard no matter what machine was built to solve them. But in the latter part of the 20^{th} century, an intriguing counterexample emerged: quantum mechanics. The simulation of straightforward quantum systems seemed to have an exponential computational cost. This led to a provocative question: what if one were to build a computer from quantum mechanical components? Could that machine outperform the Turing machine, and efficiently simulate arbitrary quantum processes? And are there other hard problems that such a machine could efficiently solve?

In this dissertation we explore several questions stemming from those ideas. First, classical simulation of quantum many-body physics may be hard, but modern supercomputers are extremely powerful—with cutting-edge innovations in both hardware and algorithms, what quantum simulations can be achieved, and what physics can we learn from them? Second, while there has been astounding progress in the development of quantum computers, they are still small and noisy—what can we do on these near-term devices, that cannot be done with the powerful classical supercomputers just described? Furthermore, if we do a quantum mechanical computation that seems to be infeasible for even the fastest classical machines, how do we check that the result is actually *correct*? Answering these questions requires deeply exploring the physical nature of computing; at heart, it comes down to the beautiful puzzle of organizing the physical world around us to process information.

In memory of Maddie Dickens, who didn't make it to graduation with us.
May you rest in peace. We miss you.

Contents

List of Figures

List of Tables

Acknowledgments

So many people have helped shape me into the scientist and person that I am today, that it is a hopeless prospect to thank them all here. But I will begin by thanking the people who started it all off, and have continued tirelessly supporting me through today: my parents. The ways I owe a debt of gratitude to them are innumerable; here I will try to list at least some that are directly relevant to my academic life. I thank my mom for showing me the wonder of the natural world, whether in opening insect galls to look at the larvae inside, or examining the traces of Vermont's natural history in ammonite fossils. I also thank her for teaching me that while academics are important, skipping school can be just as important (especially when there's 8 inches of fresh snow on the ski slopes). I thank my dad for teaching me how to solder, and more generally to build things whether electronic or not—and for the lesson that many things can be fixed or even made into something better if you understand how they work (and sometimes, even if you don't!). I also thank him for installing Linux on a desktop that was going to be thrown away and leaving it in my room when I was a kid, which kicked off a lifelong journey of fiddling with computers. To both my parents, I extend my thanks for expressing their pride in my accomplishments, but making sure I never got *too* proud of myself. It is hard to put in words how fortunate and privileged I feel for the opportunities I had growing up, and throughout my life to the present day.

On the topic of family, I would also like to thank my older sister Kate, for being someone I can always call for any reason—whether to talk through any problem, or just to play zombie video games over the internet. You have been an amazing role model my whole life. When we were kids you forged your way through each level of school and taught me invaluable lessons that helped me succeed there a year later (although I still somehow got slightly worse grades than you). As an adult you have shown me that it is possible to materially improve the world while doing the things you love. I feel so fortunate to have a sibling who is also a best friend.

On the academic side, I have to thank the many people who have supported me throughout my (many) years of school. From my time working at CERN as an undergraduate, I thank Till Eifert and Ben Nachman for teaching me a wide array of things, paramount of which was how to think like a scientist. I also appreciate them for staying patient with me when I hadn't quite figured science out yet (and when I broke the entire SLAC supercomputer cluster). I also owe many thanks to my fellow student researchers and dear friends Will DeRocco and Field Rogers, for our many adventures scientific and otherwise, and for taking care of me and helping me make (reasonably) good decisions before my frontal lobe was fully formed. I am so glad we somewhat spontaneously decided to live together that summer; it began a friendship I will cherish and care for always. I am excited to finally join you two as a PhD in Physics, and always appreciate your insights into the crazy journey of academia (and life). Will, I have you specifically to thank for making a somewhat offhand comment our third year of college: "You really like quantum mechanics and computers—have you ever considered studying quantum computing?" You probably didn't realize the direct role you played in this dissertation's existence!

On that topic, I would like to thank Prof. Liang Jiang, my first advisor in quantum computing, and (now Prof.) Stefan Krastanov, at the time a graduate student, with whom I worked closely. You two showed me the joy of theory research, and always had faith in me despite my considerable lack of background knowledge as an undergraduate. You also showed me that a lot of scientific progress can be made by essentially just hanging out, goofing around, and throwing around ideas. The fun I had working on quantum computing with you was one of my main motivations for pursuing a PhD. Liang, I also want to thank you for introducing me to the person who ultimately would become my PhD advisor (and for nudging him to respond when he didn't get back to my first email!).

That person was Prof. Norman Yao, who has guided and supported me endlessly through the extraordinary journey that is a PhD. Norm, your genuine joy and excitement in doing physics is infectious. Among the many, many lessons you taught me over the past six years, one that stands out the most is that science is a social, collaborative pursuit as much as a technical one. It is almost comical how frequently I introduce myself to someone new and hear, "Oh, you work with Norm Yao? He's an old friend." Early in my time working with you, I somewhat tentatively suggested publishing as open source the software package that ultimately became `dynamite` (Chapter 2 of this dissertation), so that everyone in our field could use it. At the time I didn't know you as well, and I wasn't sure how you would react to giving away this tool to the "competition." Your response was one of near incredulity—at the prospect of *not* sharing it with the community! Working on other projects in later years, whenever we got any hint that another research group was working on something similar to us, your first instinct was always to start a collaboration with them, rather than to compete. That attitude led to a research experience that was both more productive, and much more fun.

Norm, I also deeply appreciate you taking my "picky" research tastes in stride. A year or two into graduate school, I walked into your office feeling *very* nervous, to tell you that I simply was not enjoying the projects I was working on at the time, and that I wanted to change my research direction dramatically. You told me that even though you weren't an expert (yet) in what I wanted to study, you were more than happy to embark on that journey with me, and that we could learn together along the way. I walked out of your office with all my anxieties melted away. I think that level of flexibility is quite rare among academic advisors, and I certainly do not take it for granted. Finally, I want to thank you for your support in all forms throughout all these years. Whenever I have a question, sometimes about physics but even more frequently about how to navigate academic life, you have been always been an invaluable help. I look forward to continuing to stay in touch, and I am certain that our scientific collaboration will continue (and not just because we need to publish these papers!). I'll make sure to visit whenever I'm in town.

There are a number of other people at UC Berkeley to whom I owe thanks as well. I thank Prof. Umesh Vazirani and Prof. Joel Moore for serving on my dissertation committee, as well as Prof. Hartmut Häffner for serving on my qualifying examination committee. More broadly, I thank these professors for interesting chats over lunch or at department events. I thank Umesh in particular for welcoming me into the theoretical computer science community

at Berkeley, and for always having a keen scientific insight or interesting story to share.

I also thank my fellow graduate students at Berkeley for being the ones that truly have made the physics department into a community instead of just a place of work. So many of you work tirelessly to make this institution more equitable and just in the face of an academic culture that sometimes feels like it has infinite inertia, not to mention impenetrable bureaucracy. To all the members of UAW 2865, it was an honor to go on strike with you and fight for a contract that will help make graduate school at the University of California more accessible to everyone, not just those with outside financial and other support. Solidarity forever. To all my friends in IGenSpectrum, it has been a joy creating community together, and I am very thankful to know each of you beautiful people. Keep being your true selves and please keep in touch. To everyone in the Yao lab, thanks for being great colleagues and friends, and teaching me all kinds of interesting things, about physics but also about the world.

Finally, I thank my spouse, Sara Kahanamoku-Meyer. I honestly don't know where, or even who, I would be without you. I learn new things from you every day, and after over 10 years it is really starting to add up. (Imagine where I will be after several *more* decades of spending time with you!) Graduate school was very hard for both of us, but figuring it out together made the worst parts manageable, and the rest a joy. Whether you were giving me (or, that one time, Norm) advice about statistical methods, or taking care of our dog during the long days of writing this dissertation, or bringing me and our picnic blanket to the Berkeley Marina to drink wine and look at the Bay, you truly got me through this PhD. You are an amazing scientist, but more importantly than that, you give your whole self to the work of applying your immense knowledge and skills to better the communities you come from. I can't wait to see what you accomplish next, and am excited to continue the journey with you.

Citations to previously published work

Most of the work presented in this dissertation is adapted from work that has been previously published, or is in preparation for publishing. Here I cite the various works from which each chapter's contents are adapted.

The main content of Chapter 2 is in preparation for submission to a journal as the first work listed below. It also includes content from three other works:

> G. D. Kahanamoku-Meyer, J. Wei, and N. Y. Yao, "`dynamite`: massively parallel numerics for many-body quantum spin systems."
>
> F. Machado, G. D. Kahanamoku-Meyer, D. V. Else, *et al.*, "Exponentially slow heating in short and long-range interacting Floquet systems," *Physical Review Research*, vol. 1, no. 3, p. 033 202, Dec. 2019, Publisher: American Physical Society, DOI: 10.1103/PhysRevResearch.1.033202
>
> F. Machado, D. V. Else, G. D. Kahanamoku-Meyer, *et al.*, "Long-Range Prethermal Phases of Nonequilibrium Matter," *Physical Review X*, vol. 10, no. 1, p. 011 043, Feb. 2020, DOI: 10.1103/PhysRevX.10.011043
>
> B. Kobrin, Z. Yang, G. D. Kahanamoku-Meyer, *et al.*, "Many-Body Chaos in the Sachdev-Ye-Kitaev Model," *Physical Review Letters*, vol. 126, no. 3, p. 030 602, Jan. 2021, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.126.030602

Chapter 3 is adapted from:

> R. Van Beeumen, G. D. Kahanamoku-Meyer, N. Y. Yao, *et al.*, "A Scalable Matrix-Free Iterative Eigensolver for Studying Many-Body Localization," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, ser. HPCAsia2020, Fukuoka, Japan: Association for Computing Machinery, Jan. 2020, pp. 179–187, ISBN: 978-1-4503-7236-7, DOI: 10.1145/3368474.3368497

Chapter 4 is adapted from the following preprint, currently in review at *Quantum*:

G. D. Kahanamoku-Meyer, "Forging quantum data: Classically defeating an IQP-based quantum test," *arXiv:1912.05547 [quant-ph]*, Dec. 2019, [Online]. Available: `http://arxiv.org/abs/1912.05547` (visited on 02/14/2020)

Chapter 5 is adapted from the first work listed below. It also includes some content from the preprint listed after that, which will appear at *CRYPTO 2023*:

G. D. Kahanamoku-Meyer, S. Choi, U. V. Vazirani, *et al.*, "Classically-Verifiable Quantum Advantage from a Computational Bell Test," *Nature Physics*, vol. 18, no. 8, pp. 918–924, Aug. 2022, arXiv:2104.00687 [quant-ph], ISSN: 1745-2473, 1745-2481, DOI: `10.1038/s41567-022-01643-7`

Z. Brakerski, A. Gheorghiu, G. D. Kahanamoku-Meyer, *et al.*, *Simple Tests of Quantumness Also Certify Qubits*, arXiv:2303.01293 [quant-ph], Mar. 2023, DOI: `10.48550/arXiv.2303.01293`

Chapter 6 is adapted from the following preprint, which has been accepted at *Nature Physics*:

D. Zhu, G. D. Kahanamoku-Meyer, L. Lewis, *et al.*, "Interactive Protocols for Classically-Verifiable Quantum Advantage," *arXiv:2112.05156 [cond-mat, physics:quant-ph]*, Dec. 2021, arXiv: 2112.05156, [Online]. Available: `http://arxiv.org/abs/2112.05156` (visited on 01/26/2022)

Chapter 7 is in preparation for submission to a journal as:

G. D. Kahanamoku-Meyer and N. Y. Yao, "Quantum fast multiplication with minimal overhead."

Chapter 1

Introduction

All computers are just carefully organized sand.
— Randall Munroe, *xkcd.com/1349*

In the lobby of the Stata Center at MIT sits a wooden contraption that has an appearance somewhat akin to an oversized pinball machine. The sloped surface contains a number of barriers, holes, and movable toggles that can be set to the left or the right. In a slot at the top sit a set of billiard balls. Passersby may find a clue to the device's purpose in the labels next to some of the toggles, such as "COUNT," "MULTIPLY," and "MEMORY REGISTER." It is a mechanical calculator called the DIGI-COMP II. By flipping the toggles one can input numbers and choose an arithmetic operation to perform. With the computation set, the user releases a billiard ball down the slope. As it weaves its way through the various elements on the surface of the board the ball will flip various toggles, until it reaches the bottom and causes another ball to release from the top. This next ball will follow a new path due to the action of the previous one, flipping yet more toggles and eventually releasing yet another ball. Eventually the position of the toggles will be such that new balls will stop rolling down the slope, and the operation will halt. In what may seem a miracle to the uninitiated, the user will find that the toggles now show the mathematical result—the sum, difference, product, or quotient of the numbers that were input at the start!

While the physical nature of computing is particularly obvious in the DIGI-COMP II, this is broadly how *all* computers work. Most of them use electrons instead of billiard balls, but the general principle of operation is the same: data is input via the physical manipulation of part of the device and then the physics of the system evolves forward in time, ultimately resulting in a physical output that can be interpreted by a human (or perhaps cause some other useful effect in the world). Of course, in modern times, these physical processes are easy to forget, or at least ignore: the size of the smallest features in semiconductor chips being manufactured in 2023 is a few nanometers—roughly the diameter of the helical structure of DNA! Yet even if we do not think about it, we all still intuitively know that computations are physical processes happening inside the devices we use every day—a fact that somehow feels simultaneously obvious and surprising. When watching a high-resolution video, we may find

that our device gets hot, as energy gets expended by the operations required to update each of the millions of pixels 60 times every second. If we let the video play for long enough, we may need to recharge the device's battery by physically connecting it to a source of energy, which otherwise could power a vacuum cleaner or a lamp. More directly, if we put a photo album on a USB thumb drive, we know the photos are somehow *in* it, in a very physical sense—if we carry the drive to a friend's house, or send it through the mail, or throw it off the Golden Gate Bridge, the photos go with it. Ultimately, computing is physics, and the task of building (and using) computers corresponds to applying the laws of physics to very carefully organize the matter around us in such a way that it can encode and manipulate information.

Hiding the physical nature of computing has been crucially important, however, for facilitating the development of the diversity of computing devices that surround us today. When we open an email, or tap an Instagram ad for a sunflower-shaped dog costume, the abstract result should not depend on the specific physical processes that occur. On any of the myriad models of smartphone, tablet, laptop, or whatever else, which at a low level may operate with dramatically different computational building blocks, we will see the email's contents, or have 30 fewer dollars in our bank accounts, respectively. This abstraction applies even for most software developers: the advent of compilers in the mid-20th century created a separation between the creation of programs and their execution, allowing code to be written in a universal language like Fortran or C, and only later compiled into the specific, individual operations used by a particular processor. It is not an exaggeration to say that modern computing could not exist without it.

The intellectual foundations of this idea were laid by a brilliant insight of Alan Turing and others in the 1930s. Turing proposed a simple abstract model for a physical system that can compute any function that is computable *at all*—a device soon termed a "Turing machine."[1] This led to the concept of *Turing completeness*: given a computational system, it is Turing complete if it can be configured to simulate a Turing machine, which implies that it can compute any computable function. Over the succeeding decades, this idea was extended to explore not only which functions can be computed, but how efficiently it can be done. Astoundingly, computer scientists found that no matter how they designed a computer (within the laws of physics), Turing's simple, abstract construction could simulate it efficiently. Viewed from another perspective, they found that the set of mathematical problems that are possible but *inefficient* for a Turing machine to compute remain stubbornly inefficient no matter what computational system is used. This idea has been termed the *extended Church-Turing thesis*. [1]

However, hints that this may not represent the full story began to emerge near the end of the 20$^{\text{th}}$ century, when scientists began to consider the prospects of simulating quantum mechanics using computers. In a foundational talk in 1981, Feynman pointed out that

[1]To be precise, this depends on one's definition of "the set of computable functions;" Turing showed that the set of functions his machine could compute was equal to the that of two other contemporaneous definitions of computability. In modern theoretical computer science, computability is in fact usually defined *in terms of* Turing machines.

unlike in the simulation of classical physics, fully describing the state of a quantum system seems to require manipulating an *exponential* number of values—a task that certainly is not efficient. He proposed that instead of using standard computing hardware, perhaps such a simulation could be performed efficiently on a new type of computer, itself built from *quantum mechanical* parts. Such a device would constitute a clear violation of the extended Church-Turing thesis. This led to an extremely provocative further question: are there *other* problems, not directly related to quantum mechanics, that are hard for Turing machines but can be computed efficiently using a machine built from quantum mechanics? Just 13 years later, Peter Shor stunned the world of computer science by giving a fast quantum algorithm for factoring numbers—a computational problem that had been the target of efforts by number theorists for millennia. (In fact, factoring, and the related discrete logarithm problem, were considered to be hard with such certainty that their hardness has been used as the backbone of much of digital security. This fact has played a large part in the interest of national governments in quantum computing.) After Shor's work, research efforts in the new field of quantum computing increased dramatically, both with intense experimental work to physically construct such devices and theoretical efforts to develop new algorithms that could run on them.[2] Yet despite the gargantuan efforts made over the past few decades, quantum computing remains stubbornly difficult. As of yet, the physical quantum computers that have been constructed are only beginning to pass the cusp of what can be simulated with modern classical supercomputers, and even so, while a wide array of theoretical applications have been proposed, finding realistic and useful problems for which these first small, noisy quantum machines can meaningfully improve on the performance of classical computers has remained elusive. To put things in perspective, the current record for integer factorization via Shor's algorithm seems to be the factorization of 21 as 7×3 in 2012, [4] but even that record had its validity soon called into question [5].[3]

As a result of all of this, we find ourselves at an exciting time for quantum science. In terms of looking for new physics in many-body quantum systems, we are at a juncture where modern supercomputers and cutting-edge experiments can simulate quantum systems of roughly similar complexity. Each technique has its own strengths, and the ability to look for new physical phenomena in numerical simulations and then explore them further with experiments, or vice versa, has proven extremely powerful, leading to the discovery of new phases of matter and new insights into quantum materials. Encouragingly too, both experiments and numerics have improved rapidly over the past few decades. On the quantum computing side, despite the current lack of a "killer application" for the near term, the astounding progress in the precise control of quantum systems holds promise. It has been

[2]For in-depth yet personal discussions of the history of quantum computing, we recommend the recent retrospectives by Peter Shor and John Preskill. [2], [3]

[3]There are so-called "variational" quantum factoring algorithms which have been used to demonstrate factoring of somewhat larger numbers than 21 on near-term quantum devices. However, these algorithms become inefficient rapidly as the numbers get larger and would not be useful at all for factoring numbers larger than what can already be done with classical computers, even if a large error-free quantum computer could be constructed.

said that when algorithms can only run in one's imagination, it is very difficult to develop them; we are finally reaching the point where those designing quantum algorithms can move off the blackboard and play with real quantum devices. With any luck, this will lead to the exploration of new directions that we have not yet imagined.

This dissertation focuses on the implications of the difference in computational power of classical and quantum systems, and how they can interact with each other. For the rest of this chapter, we give an overview of some of the foundational ideas upon which this work relies. The discussion is targeted at a reader who is knowledgeable of physics, but perhaps new to the subfields being discussed. It does not endeavor to go deeply into technical details, but instead to give a broad intuitive landscape of the techniques currently being used to make progress in these fields. Hopefully this can serve as a resource for early graduate students looking to study these topics.

After the introduction, this dissertation is divided into two parts. In the first part, we examine how, despite the complexity-theoretic hardness of classically simulating quantum mechanics, the tools of modern supercomputing can be used to simulate quantum systems of sufficiently large size that meaningful scientific insight can be produced. In Chapter 2, we present a numerical library called `dynamite`, which has a simple and intuitive programmatic interface for performing numerical simulations of many-body quantum systems, but is very powerful, with the capability to accelerate the computations via massive parallelism on supercomputers and graphics processing units (GPUs). In Chapter 3 we present the implementation of algorithmic and practical innovations targeting a specific many-body physics problem, that of "many-body localization" (MBL). Our approach dramatically reduces the memory usage compared to previous works, enabling the analysis of larger physical systems with less resources. In an amusing connection to the physicality of computing, for both these chapters the work required moving past the compiler abstractions described earlier, instead incorporating knowledge of the physical characteristics and low-level operations of the compute hardware upon which the code would run in order to eke out as much performance as possible.

In the second part, we focus on a subtlety of the ongoing race to build quantum computers: if a quantum device claims to be able to perform computations that no classical computer can, how can we *verify* that it actually has done so? What if we do not trust the quantum device, either because its behavior is not well characterized, or because we do not trust the humans behind it? In Chapter 4, we examine a proposal from 2008 for a "proof of quantumness" by which a quantum *prover* can convince a skeptical classical *verifier* of its capability to perform computations that would be infeasible classically. We break the protocol, showing that there in fact exists an algorithm by which a classical cheater could reproduce the behavior of an honest quantum prover in the protocol, which destroys the test's guarantee that the prover is quantum. In Chapter 5 we propose a new proof of quantumness protocol in which classical cheating is *provably* as hard as factoring integers, avoiding the pitfalls described in the previous chapter. The new protocol can be implemented with less quantum resources than Shor's algorithm, yielding potential for its real-world use in the time before quantum computers are capable of running algorithms as complex as Shor's. In

Chapter 6, we present the results of a first, small-scale demonstration of that proof of quantumness protocol (and another related protocol) on an ion-trap quantum computer. The results represent a technological step forward, as they involve performing measurements in the middle of a set of quantum operations and then continuing operation afterwards. These mid-circuit measurements have historically proven to be extremely technically challenging to implement, and open the door to important new paradigms of quantum computing such a real-time error correction. In Chapter 7, we present a novel construction for quantum circuits implementing integer multiplication—a key ingredient for both the proofs of quantumness just described, and for Shor's algorithm itself. We show that the math behind fast multiplication algorithms which have been known for decades in the classical setting can be applied to an inherently quantum multiplication method which uses the phases of quantum states to perform arithmetic. Furthermore we show that a considerable fraction of the work can be performed in classical pre-computation when the quantum circuits are being compiled. The construction dramatically reduces the number of extra qubits used, while remaining competitive in the number fo quantum operations required. Finally, in Chapter 8, we give concluding remarks and look towards the future.

1.1 Numerical studies of many-body quantum systems

The study of many-body physics has been known for centuries to be challenging, even in the classical setting. Long before the advent of quantum mechanics, in studying the collective motion of the Sun, Moon, and Earth, physicists found (and eventually proved) that the dynamics of just *three* masses interacting via Newtonian gravitation did not have a general closed form solution. Worse, many of the physical systems relevant to our everyday lives have far more than 3 particles—a teaspoon of water contains roughly 10^{23} molecules! Clearly, directly solving for the dynamics of each individual molecule is out of the question.

Until the mid- to late-20^{th} century, there were only two main strategies available for treating many-body systems. The first is to find specific cases that *are* solvable, and then leverage them to find approximate "nearby" solutions as well. (For the three-body orbit problem, an example of this is approximating one of the bodies as having zero mass in comparison to the other two, which is frequently appropriate in real astrophysical scenarios.) The second, which is appropriate for systems of very many particles and is used in the fields of statistical and fluid mechanics, is to not focus on the dynamics of each individual particle but to instead find a description of their *collective* behavior as a whole. These tools allowed physicists to make an astounding amount of progress, but certain questions remained impervious to these methods.

In the past several decades, a powerful new tool has come onto the scene: modern computers, capable of performing billions of mathematical operations per second, with which the dynamics of many interacting particles can be computed numerically! In the classical case, thousands or even millions of particles can be simulated at once, providing a good approximation of the thermodynamic limit of infinite system size. Quantum many-body

physics, however, presents yet another challenge: the exponential size of the Hilbert space as a function of the number of particles. Consider of a collection of L quantum particles, each with just a two-dimensional local Hilbert space (for example, spin-1/2 particles without any other degrees of freedom). Their collective wavefunction is a vector[4] in a Hilbert space of dimension 2^L—over 1,000 for a collection of just 10 particles, and over 1,000,000 for 20 particles. This is the practical manifestation of the exponential complexity of classically simulating quantum mechanics discussed earlier in the introduction: informally, the cost of simulating 1,000,000 classical particles can be compared to the cost of simulating just 20 quantum ones.

Considering this fact, and Feynman's point that quantum computers may be better suited for simulating quantum mechanics than classical ones, one might ask why classical simulation of quantum mechanics is worth it in the first place. There are three main benefits, by which numerical study has proven invaluable in the study of quantum many-body physics. First, a classical simulation allows one to interrogate the behavior of a quantum system under ideal conditions. The noise that is ever-present in experiments (and has particularly been the bane of those attempting to build quantum computers) can simply be turned off in a simulation. This can be hugely helpful in assessing whether an observed effect is real or simply due to imperfections of the experiment. Secondly, the development process is usually much faster. Depending on the software used for simulation, it can be possible to arbitrarily adjust the interactions between particles, or the physical geometry of the system, or any of a number of other parameters, at the press of a button—adjustments which may require considerable effort to implement in an experiment, if they are possible at all. (Indeed, the goal of the software library we present in Chapter 2 is precisely to make such simulations as easy and quick as possible!) Finally, and most importantly, classical simulation gives access to data which simply is inaccessible in a real quantum experiment. Performing measurements on quantum states causes wavefunction collapse—by Holevo's theorem, even though L spin-1/2 particles have a state vector of dimension 2^L, the number of classical bits of information that can be extracted by measuring that wavefunction is only L! Meanwhile, in a classical simulation, arbitrary functions of the state vector can be computed. This is particularly important for "non-observable" quantities such as the entanglement entropy of a state, which in an experiment cannot always be estimated with good statistics without performing an exponential number of trials.

1.1.1 How to represent a quantum state in a classical computer

Having hopefully convinced the reader of the merits of pursuing classical simulation of quantum many-body physics, as difficult as it may be, we now move on to discussing the first necessary step to performing numerical simulations: representing a quantum state on a classical computer. Much work has been done exploring various strategies for this. Here we discuss two that are most commonly used: storing the state vector explicitly as an array of

[4]Technically a ray, since the normalization is not physically important.

complex numbers, and tensor network methods. We note that there exist a wide range of other techniques whose use is somewhat less widespread. These include, for example, stabilizer states, which can efficiently represent states generated by quantum circuits of Clifford gates (and "nearby" states) [6], [7]; and neural network states, in which a neural network is trained to produce the coefficients of the state vector [8], [9].

In the below, we will consider quantum systems consisting of L two-level systems. The Hilbert space is mathematically represented by the space $\mathbb{C}_2^{\otimes L}$. We consider a pure quantum state $|\psi\rangle$ on this Hilbert space, that we desire to represent numerically. We do not cover here the various techniques for numerically representing and computing with mixed states; however, we note that models with noise can be simulated with pure states, for example via the formalism of "quantum trajectories." [10]

1.1.1.1 Vectors of coefficients

The most straightforward way to represent such a state numerically is by picking a set of basis states $|\phi_i\rangle$ that span the Hilbert space, and storing an array of complex numbers c_i such that $|\psi\rangle = \sum_i c_i |\phi_i\rangle$. The obvious challenge with this strategy is that the number of c_i that must be stored is equal to the dimension of the Hilbert space, which is exponential in L. To give a sense of scale, with 1 Terabyte of RAM (achievable with a few nodes of a modern computer cluster), and complex numbers stored as a pair of 8 byte floating point numbers, one can represent a system of $\log_2(1 \text{ Terabyte}/16 \text{ bytes}) \approx 35$ spins—not a huge number, but certainly large enough to see many-body collective behavior in certain systems. Despite the limitation to moderate system sizes, the power of this representation is that it can describe entirely *arbitrary* quantum states. This benefit is often worth the exponential cost—indeed, this is how states are represented in the numerical work of Chapters 2 and 3.

From an information-theoretic perspective, an exponential number of classical bits is simply *required* to represent arbitrary quantum states, because the space of quantum states is exponentially large. Fortunately, in many physical situations, the states of interest are likely to not be entirely arbitrary but to belong to some subspace of the larger Hilbert space, and if we can find a more efficient way of representing the states in that subspace it becomes possible to handle larger system sizes. One application of this idea, which is used widely, is to use conservation laws to divide the Hilbert space into the direct sum of several smaller subspaces. As an example, the Hamiltonian governing the dynamics of a system of quantum spins might conserve their total magnetization in a particular direction. In that case we can ensure that we choose a set of basis states for which that magnetization operator is diagonal, and break the Hilbert space into "sectors" each with a different magnetization. In this case, for spin 1/2 particles with zero total magnetization, our 1 Terabyte of RAM can store the state vector of 38 spins with the total magnetization set to zero, as opposed to the 35 spins that were possible in the general case. (The zero magnetization subspace has the largest dimension; for different values things improve even more.) Note that this specific conservation law is implemented in both Chapters 2 and 3; a number of other conservation

laws are frequently used in physics studies and several more are implemented in the `dynamite` package presented in Chapter 2.

Even with conservation laws that allow us to ignore a large fraction of the Hilbert space, the number of coefficients that must be stored usually remains superpolynomial in the system size. We now move on to discussing a way of representing quantum states using only a *polynomial* amount of data—so called "tensor networks."

1.1.1.2 Tensor networks

In the previous section, we reduced the size of the effective Hilbert space by only considering states with a given value of a conserved quantity; the intuition behind tensor network representations of quantum states is to focus only on states with low entanglement. This idea is well-motivated because the states encountered in physics studies frequently have this property—for example, this is the case for the ground states of a large class of Hamiltonians. The challenge, of course, is to find a way of representing low-entanglement states such that they can be stored, and manipulated, efficiently. Tensor networks attempt to do just that.

A number of extensive, pedagogical introductions to tensor network methods have been written to which I would direct any reader interested in deeply exploring this topic; [11]–[13] instead of creating yet another (probably of worse quality) I instead here will give a high-level overview of their structure, as I like to view it. My hope is that it can be helpful in building intuition for those whose mathematical style is similar to my own.

The broad idea is based off of the Schmidt decomposition. Consider a quantum system with two parts, which we denote A and B, with local Hilbert spaces H_A and H_B of dimension n and m respectively. In general, we may write the global state of the system in terms of any orthogonal sets of basis vectors $\{|a_i\rangle\}$ and $\{|b_j\rangle\}$, as $|\psi\rangle = \sum_{ij} c_{ij} |a_i\rangle |b_j\rangle$. The power of the Schmidt decomposition is that it shows how to construct particular orthonormal bases $\{|\alpha_i\rangle\}$ and $\{|\beta_i\rangle\}$, and coefficients s_i, such that the sum only needs to run over a single index:

$$|\psi\rangle = \sum_i s_i |\alpha_i\rangle |\beta_i\rangle \tag{1.1}$$

Viewed another way, it finds a basis for H_A and H_B such that the c_{ij} are only nonzero when $i = j$. At first this seems too good to be true: we are representing a quantum state on the global Hilbert space, which has dimension mn, using only $\min(m, n)$ coefficients! Alas, there is no free lunch; the trick is that the basis vectors $\{|\alpha_i\rangle\}$ and $\{|\beta_i\rangle\}$ are themselves dependent on $|\psi\rangle$, and so must be stored explicitly. The real benefit of the Schmidt decomposition comes when we consider *entanglement*.

Recall the definition of the von Neumann entanglement entropy of subsystem A, denoted as S_A. It is a function of $\rho_A = \mathrm{Tr}_B |\psi\rangle$, the reduced density matrix of subsystem A when B has been removed via a partial trace.

$$S_A = -\mathrm{Tr}\left[\rho_A \log \rho_A\right] \tag{1.2}$$

It is straightforward to see from the definition of the partial trace, and the orthonormality of the basis vectors $\{|\alpha_i\rangle\}$ and $\{|\beta_i\rangle\}$, that the Schmidt decomposition diagonalizes the density matrix:

$$\rho_A = \sum_i s_i^2 \, |\alpha_i\rangle \langle\alpha_i| \tag{1.3}$$

and thus the entanglement entropy can be written straightforwardly in terms of the s_i:

$$S_A = -\sum_i s_i^2 \log(s_i^2) \tag{1.4}$$

Careful inspection of this expression, combined with the fact that $\sum_i s_i^2 = 1$, yields a powerful fact: informally, if the entanglement entropy S_A is small, then only a few of the s_i are non-negligible! This suggests the following approximation: for some cutoff ϵ, simply drop the terms of Eq. 1.1 for which $s_i < \epsilon$.

$$|\psi\rangle \approx \sum_{\substack{i \\ s_i \geq \epsilon}} s_i \, |\alpha_i\rangle \, |\beta_i\rangle \tag{1.5}$$

If most of the s_i are smaller than ϵ, we may only need to store a small number of tuples $(s_i, |\alpha_i\rangle, |\beta_i\rangle)$. The intuition here is really nice: if there is *no* entanglement between the two parts, then the state is trivially the tensor product of a pure state on each part: $|\psi\rangle = |\psi_A\rangle \, |\psi_B\rangle$. The Schmidt decomposition allows us to see that this is the most extreme case of a more general fact, that low-entanglement states are well approximated by a linear combination of just a few tensor products of that form.

Let's now look at how this can be applied to our system of L two-level spins, arranged in a 1D chain. Let system A be the leftmost spin, and system B be the remainder of the chain. Since H_A has dimension two, applying the Schmidt decomposition, we will get two tuples $(s_i, |\alpha_i\rangle, |\beta_i\rangle)$ where the two $|\alpha_i\rangle$ are vectors each of dimension 2 and the two $|\beta_i\rangle$ are vectors of dimension 2^{L-1}. The key to making this useful is that we may now apply the Schmidt decomposition *again*, to the vectors $|\beta_i\rangle$! Letting our new subsystem A' be the second-to-leftmost spin and B' be the remaining spins on the right, we now get a total of 4 tuples $(s'_i, |\alpha'_i\rangle, |\beta'_i\rangle)$—two from each of the two $|\beta_i\rangle$. The $|\alpha_i\rangle$ are once again each of dimension 2, and the $|\beta_i\rangle$ are now of dimension 2^{L-2}. We may continue this plan across the entire chain of spins, ultimately decomposing our state $|\psi\rangle$ into a collection of sets of basis vectors, each of dimension 2. The benefit is obvious—we are never storing any vectors larger than dimension 2! The downside is that without any truncation, the *number* of such basis vectors grows exponentially with the distance from the end of the spin chain. However as discussed above, for states with low entanglement, we can drop most of the vectors since their associated s_i are small. In fact, for states without extensive entanglement, we need only keep a *constant* number of vectors on each spin to achieve a good approximation of ψ. This constant is called the "bond dimension" and is usually denoted by χ.

The construction just described is called a matrix product state (MPS). The standard way of viewing it is as a *tensor network*: a set of tensors, each one representing the set of

basis vectors associated with each spin, with "bonds" between them corresponding to the shared indices i in the sum of Eq. 1.1. It turns out the linear chain of tensors we have described so far is just one of a large class of methods for representing quantum states via tensor networks. This has yielded new ways of representing states with different physical geometry of the interactions, and different entanglement structures—well-known examples include Projected Entangled Pair States (PEPS) and Multiscale Entanglement Renormalization Ansatz (MERA). [13], [14] Moving past the case of a 1D physical system is a challenging pursuit, with surprising pitfalls such as constructions that can efficiently represent certain quantum states accurately, but for which actually computing any quantities of interest is exponentially computationally hard! This is an active area of research, and we direct the interested reader to any of the extensive reviews on the subject. [11]–[13]

1.1.2 Simulating quantum dynamics

Having discussed various ways of representing quantum states on classical computers, we now turn to performing computations on them. The most obvious operation of interest is the numerical simulation of their evolution through time. Consider a quantum system with some Hamiltonian $H(t)$ (with the argument making explicit that the Hamiltonian may depend on time). This time evolution is represented mathematically by the time evolution unitary $U(t, t_0)$ that corresponds to the solution to the time-dependent Schrodinger equation

$$\frac{\partial}{\partial t}U(t, t_0) = H(t)U(t, t_0) \tag{1.6}$$

For a time-independent Hamiltonian, the solution has a straightforward form:

$$U(t, t_0) = e^{-iH(t-t_0)} \tag{1.7}$$

For a time-dependent Hamiltonian, things are a bit more complicated. A classic mistake made by those new to the field is to write $U(t, t_0)$ as

$$U(t, t_0) = e^{-i\int_{t'=t_0}^{t} H(t')dt'}$$

which at first glance looks great—the time evolution should correspond to the cumulative effect of the Hamiltonian acting from time t_0 to time t. (It feels like a rite of passage to screw this up! I certainly did at least once early in my research.) But it is incorrect, which can be seen as follows. Consider a simple time-dependent Hamiltonian which consists of a static Hamiltonian H_A acting for time t_A followed by another static Hamiltonian H_B acting for time t_B. The evolution over one cycle of time $t_A + t_B$ is (correctly) described by the unitary

$$U_{AB} = e^{-iH_B t_B}e^{-iH_A t_A} \tag{1.8}$$

which is crucially *not* necessarily equal to $e^{-i(H_B t_B + H_A t_A)}$—that equality only holds if H_A and H_B commute! To account for the fact that $H(t)$ may not commute with itself for all

times t, one must use what is called the *time-ordered exponential*, which is denoted thus:

$$U(t, t_0) = \mathcal{T}\{e^{-i\int_{t'=t_0}^{t} H(t')dt'}\} \tag{1.9}$$

The precise definition and use of the time-ordered exponential is out of the scope of this introduction, but it turns out we will not need it for our numerical purposes: during my research, the best strategy has essentially always been to simply break up the time-dependent Hamiltonian into a piecewise time-independent one, computing each piece separately as in Equation 1.8. Thus for the rest of this section we will discuss how numerical time evolution under a static Hamiltonian H is performed.

The most straightforward way of implementing time evolution is to simply create a numerical representation of H as a matrix, and compute U via the matrix exponential (Eq. 1.7). This can be done via a matrix exponential algorithm (e.g. via the `linalg.expm()` function of the SciPy library); however, in my experience, for larger system sizes it is actually faster to solve for the eigendecomposition $H = V\Lambda V^\dagger$, where V is a matrix whose columns are the eigenvectors and Λ is the diagonal matrix of the eigenvalues. The unitary can then be computed by exponentiating Λ, which is very straightforward because it is diagonal—one simply exponentiates each eigenvalue separately. This strategy is particularly efficient if the unitary is required at many different times t, because the eigendecomposition only needs to be performed once.

The benefit of explicitly computing the unitary is that, well, one has an explicit representation of it! With that one can easily determine its action on arbitrary state vectors, or even on mixed states. Explicitly computing $\mathcal{U}$ is very costly, however: it is a square matrix of the same dimension as the Hilbert space, meaning that even just storing it requires computer memory proportional to the Hilbert space dimension *squared*—without the use of conservation laws, it becomes impractical for systems of more than just 16 spins or so. Fortunately, it is almost always overkill—the only instance I have ever encountered in which it has been necessary to explicitly compute $\mathcal{U}$ is in the study of Floquet systems, where the eigenvalues of $\mathcal{U}$ represent "quasi-energies" that are physically relevant. In virtually all other cases, what we are really interested in is the *action* of $\mathcal{U}$ on a state.

The hope that computing $\mathcal{U}|\psi\rangle$ for some state $|\psi\rangle$ may be more efficient than computing $\mathcal{U}$ in full is well-motivated if we consider the Taylor expansion of $\mathcal{U}|\psi\rangle$ in t:

$$\mathcal{U}|\psi\rangle = e^{-iHt}|\psi\rangle = |\psi\rangle - iHt|\psi\rangle - \frac{H^2t^2}{2}|\psi\rangle + \cdots \tag{1.10}$$

If t is small, the norm of each term of the expansion is exponentially smaller that the last, and we may obtain a good approximation of $\mathcal{U}|\psi\rangle$ with only a small number of terms. Importantly, it is possible to compute this expansion while only storing *three* vectors: one to hold the result, and two more in which $|\psi\rangle$, $H|\psi\rangle$, $H^2|\psi\rangle$, etc. are computed in alternating fashion. (Depending on the specifics of the situation, it may even be possible to reduce this to fewer than three, if the multiplications of the state by H can be performed in-place).

Of course, we are usually interested in time evolution for longer times t than those for which this expansion is well-controlled. In that case, we may consider the following exact decomposition

$$e^{-iHt} = \left(e^{-iHt/n}\right)^n \tag{1.11}$$

That is, we can break down the total evolution over a time t into n evolutions of time t/n. If n is sufficiently large, each of these smaller time evolutions are in the regime in which the above expansion is well-controlled, and thus the error is well-controlled across the entire evolution. This idea forms the backbone of a number of algorithms for time evolution of states, whether they are represented explicitly as a vector of coefficients, or via matrix product states.

In Time-evolving Block Decimation, an algorithm for time-evolving matrix product states, it is observed that if the expansion in Eq. 1.10 is truncated after the first order in Ht, the operators that need to be applied have support on at most two sites at once—and thus only need to be applied to at most two tensors of the matrix product state, which is efficient for matrix product states in which the bond dimension is not too large. As long as the entanglement remains small throughout the time evolution, terms of the Schmidt decomposition can continuously be truncated throughout the process (as in Eq. 1.5) while maintaining a good approximation of the state. [12]

For states stored as vectors of coefficients, as in Section 1.1.1.1, the strategy of Eq. 1.10 can be improved through the use of so-called *Krylov subspace methods*. This type of algorithm forms the backbone of the numerical package `dynamite` presented in Chapter 2, and a detailed exposition is provided there; for now, we will give the broad intuition. Instead of explicitly computing the sum Eq. 1.10, Krylov methods compute an orthonormal basis for the subspace $\text{span}\{|\psi\rangle, H|\psi\rangle, H^2|\psi\rangle, \cdots, H^{m-1}\}$ up to some cutoff m (say, 30). Then, the matrix H is projected into this small subspace and $\mathcal{U}$ is computed explicitly in the small $m \times m$ dimensional space. Using this, the vector $\mathcal{U}|\psi\rangle$ is computed and then projected back into the original Hilbert space. It is clear by inspection that this strategy will do at *least* as well as Eq. 1.10 for the same order of approximation n, but it turns out that in practice it usually does much better because it includes the parts of high-order terms of $\mathcal{U}$ that also fall in the subspace. As before, it may be useful for larger t to break the evolution down into n evolutions of a shorter time t/n, and compute each of these shorter evolutions with the method just described.

1.1.3 Eigensolving

Another operation that is widely relevant for physics studies is solving for the eigenvalues and eigenvectors of a Hamiltonian. The eigenvalues determine the energy levels of the system, and eigenstates provide insight into its physical characteristics. In many cases the ground state and first few excited states are particularly important, as they determine the system's behavior at low temperature.

Once again, the most straightforward way to find the eigenvalues and eigenvectors of a quantum operator is to numerically construct the Hamiltonian and then apply a generic matrix eigensolver (e.g. `numpy.eigh()`). But like in the time evolution case, this is simultaneously very expensive, and usually overkill. In situations where we only need the low-lying states, we can use the *variational method*. The idea is that for a given Hamiltonian H, the energy of *any* state $|\psi\rangle$ is lower-bounded by the energy of the ground state—and if we can optimize $|\psi\rangle$ to have as low an energy as possible, we can find a good approximation of the ground state energy (and hopefully of the ground state itself).

For matrix product states, the classic way of doing this is via the Density Matrix Renormalization Group (DMRG) algorithm. Informally, the idea of DMRG is to sweep back and forth across the tensors of the matrix product state, minimizing the energy at each step by locally optimizing over each one. By doing several sweeps in this way, the state becomes a better and better approximation of the true ground state; if the true ground state has low entanglement (and thus can be well-represented by a matrix product state), in practice it can usually be converged with only a few sweeps. [11]–[13]

For states stored as vectors of coefficients, we may use any of a number of optimization algorithms to attempt to find the vector $|\psi\rangle$ with the lowest value of $\langle\psi| H |\psi\rangle$. The most frequently used methods build off of the intuition of the *power method* for eigensolving. Consider a uniformly random vector, which can be written in the basis of the eigenvectors $|\phi_i\rangle$ of H (which are as of yet unknown): $|\psi_r\rangle = \sum_i c_i |\phi_i\rangle$. Observe that repeatedly multiplying this vector by H exponentially enhances the eigenvector corresponding to the largest magnitude eigenvalue: $H^m |\psi_r\rangle = \sum_i \lambda_i^m c_i |\phi_i\rangle$. By shifting the zero point of energy we may ensure that the largest magnitude eigenvalue is the most negative one, and thus this method will converge the ground state! The power method is not usually used directly, because there exist algorithms which are based on the same intuition but converge even more quickly—a class of methods called *iterative eigensolvers*. These include the Lanczos algorithm, Krylov subspace methods more generally (as described earlier for time evolution), and others. Chapters 2 and 3 make extensive use of these iterative-type eigensolvers, and detailed descriptions of how they work can be found there.

1.2 Quantum advantage

As it stands, the central concern of quantum computing is, of course, whether it can outperform classical computing—a goal termed "quantum advantage." But determining where and if quantum advantage is possible, and before that, even clearly defining it, is a surprisingly subtle pursuit. On the theoretical side, a wide array of algorithms have been discovered that solve various problems in asymptotically fewer operations on a quantum computer versus a classical one.[5] But translating these algorithms into a speedup observable in practice has

[5] In fact, this point is subtle too. Explicit proofs of a lower bound of the cost of solving a problem on a classical computer are hard to come by, so these algorithms usually correspond to speedups over the best *known* classical algorithm.

proved very difficult. A main challenge is that modern classical computers are extremely fast, in terms of the number of operations they can perform per second. Today's top supercomputers can achieve exascale performance: 10^{18}, or one quintillion, 64-bit floating point operations per second! Meanwhile, most modern quantum experiments are limited to a few hundred or thousand quantum gates *total* before noise destroys the quantum state. Even ignoring noise, the cycle time of modern quantum computers is something between 10^4 and 10^8 gates per second depending on the platform, dramatically slower than their classical counterparts.

To demonstrate a speedup in practice, the algorithmic gain from using quantum hardware must outcompete this extreme disadvantage in cycle time. It is clear that quantum algorithms which only reduce the number of operations by a polynomial amount—say requiring $\sqrt{N}$ operations when the classical computer needs N—will not close the gap without revolutionary improvements in quantum technology. Instead, near term quantum advantage requires algorithms which provide a *superpolynomial* speedup, ideally the fully exponential speedup corresponding with the classical complexity of simulating quantum mechanics itself. A few of the classic quantum algorithms, in particular Shor's, do exhibit this dramatic superpolynomial improvement over the best known classical algorithm for the same problem. Unfortunately they are quite complicated to implement, and so remain far out of reach of near-term quantum devices. For these reasons, the pursuit of experimental quantum advantage has required the devising of new computational problems and associated quantum algorithms, which are as undemanding as possible for a quantum computer yet as costly as possible for a classical one.

Note that here we will focus on quantum advantage in *computational cost*, broadly defined but with a focus on runtime. The setting to keep in mind is that of a quantum device connected to a classical computer. The classical computer sends the quantum device commands (usually the quantum gates to perform), and receives some classical data back. A weak version of the goal is that if the quantum device is replaced by a classical machine of comparable resources—say, in size, energy usage, and computation time—it should be unable to reproduce the computation performed by the quantum device. The strongest version of quantum advantage, which is the one pursued by most of the research described next, is that no classical machine, not even the world's fastest supercomputers, can reproduce the quantum device's behavior, given any practical amount of computation time and other resources.

1.2.1 The first experimental demonstrations

The problem of finding specific computational tasks for demonstrating quantum advantage in practice did not receive much attention until recently, because quantum devices were so small and noisy that they could be easily simulated by classical computers no matter what operations they performed. But in the past few years, quantum hardware has improved to the point that at least direct classical simulation has become infeasible—leading to the exploration of whether there was *some* computation, perhaps contrived and not necessarily

useful, in which a speedup could be experimentally observed. Theoretical work along these lines led to the conclusion that the most achievable way to show quantum advantage would not be to solve a problem with a deterministic answer (like "factor this integer") but instead to solve a *sampling problem*: given some data that defines a probability distribution, the task is to generate (perhaps approximately) samples from that distribution. [15]–[23] By defining the target probability distribution to correspond to the distribution of measurement results for a particular quantum state, produced by running a quantum circuit, the problem becomes very naturally suited for a quantum computer. Furthermore, intuitively, the hardness of classically reproducing the sampling results comes directly from the hardness of classically simulating generic quantum circuits. In particular, if the quantum circuit to be run has little structure (for example, if it consists of a series of random quantum gates), there should be no "shortcuts" by which a classical computer can reproduce the results, short of accurately simulating a quantum device.[6] Starting in 2019, a series of experiments were published implementing this idea at scale and marking the first quantum computations to not only outperform the top classical supercomputers, but do so to such an extent that the results could not be reproduced classically at all![7] [32]–[35]

With those results came a subtlety, however: if the output cannot be reproduced classically, how is it possible to verify that it's actually *correct*? The papers followed two parallel strategies for handling this. The first is to perform experiments in the so-called "Goldilocks zone," where finding a classical solution is very difficult but not impossible. That way, the difference in computation time (or another metric such as energy usage) can still constitute quantum advantage, yet via a large classical computation the results can be verified. The second strategy applies to computational problems past the Goldilocks zone, where direct verification is truly infeasible. In that case, experiments resorted to showing that quantum mechanical processes were the only "reasonable" explanation for the observed results. For example, in their landmark paper that began the series of experimental claims to quantum advantage, Google showed that their device performed as expected for a set of "nearby" computations that *were* possible to classically simulate, extrapolating that the device probably would perform as desired when running the classically hardest computations as well. In a similar vein, the second paper to claim quantum advantage supported their results by "ruling out alternative [classical] hypotheses." [33]

[6] While the intuition may be clear, a considerable effort was required to give theoretical backing to the hardness of random circuit sampling, and the hardness of *approximately* sampling from the distribution is still an active area of research. [24]–[31]

[7] A number of follow-up papers from other research groups demonstrated various improvements to classical techniques for more efficiently solving the sampling problems implemented in the experiments; [24]–[31] in the mean time, the quantum experiments have also improved. At this point it seems to be generally accepted that the most recent experiments are truly out of reach of classical computing.

1.2.2 Efficiently-verifiable tests

But what if there is some classical explanation that we have not considered? Or, what if we do not have any ability to look "inside" the device running our computations—for example, if we want to test the power of a quantum cloud computing service being offered over the internet? To demonstrate quantum advantage past the Goldilocks regime in these scenarios requires setting up an asymmetry in the computational problem: it should be hard to classically *solve*, but easy to classically *verify*. We can frame this idea via a structure from classical complexity theory called an interactive proof. Here, a quantum *prover* desires to demonstrate its capability to skeptical classical *verifier*. Due to this connection, this type of test has been termed a "proof of quantumness."

Shor's algorithm actually provides a straightforward protocol for achieving these goals: the classical verifier chooses two large prime numbers (in a way that would be secure for RSA encryption), multiplies them together, and sends the result to the quantum prover. If the prover can find the factors and return them, the verifier can be confident that the prover is truly quantum, to the same level of confidence that it is believed that classically factoring numbers is hard. We have already discussed why this protocol begs to be improved: the inherent challenges in running Shor's algorithm make it totally infeasible to run at scale on today's quantum devices. So, recent excitement has focused on whether the goal of creating an efficiently-verifiable "proof of quantumness" can be achieved in a way that is compatible with near-term devices.

There seem to be two direct paths towards achieving this. The first is to take a sampling problem, like the ones that were first used to show (non-efficiently-verifiable) quantum advantage, and somehow add structure to it so that the output can be efficiently verified. The second is to take a cryptographic problem, like factoring, and somehow "strip it down" to its core, such that it hopefully could require less resources than the full machinery of an algorithm like Shor's.

1.2.2.1 Adding structure to sampling problems

The challenge with the first approach is that the classical hardness of sampling problems at some level *depends* on the fact that they do not have much structure. To my knowledge, there has only been one protocol ever proposed that attempts to create a proof of quantumness in this way. It actually came long before even the non-verifiable tests of quantum advantage were proposed. In 2008, a paper was released introducing a new quantum complexity class called IQP ("Instantaneous Quantum Polynomial" time, a class of quantum circuits in which all gates commute with each other). The authors realized that the structure of IQP computations seemed to lend itself to a particular sampling problem, which could be set up such that the underlying measurement distribution frequently yielded bitstrings with a special, efficiently-checkable relationship to a secret string s that should only be known to the verifier. Since s is secret, and simulating IQP circuits is classically hard, [16], [36] only a real quantum computer should be able to reliably produce bitstrings having this special

relationship to s. The protocol remained known for over a decade, and experiments began to undertake efforts to implement it. [37] Unfortunately, it turned out that the concern raised above, that adding structure might compromise the classical hardness, was real. In 2019 I found a classical algorithm by which the secret string s can be recovered in its entirety, destroying the classical hardness claim of the protocol. The original protocol, and the algorithm to break it, are described in Chapter 4.

1.2.2.2 Protocols based on cryptography

While that first approach does not seem to have led to any further advances, progress has been made on the second—simplifying cryptographic problems to make them more feasible. This may initially seem surprising, considering that "make factoring easier for near term devices" is an obvious and intensely sought-after goal in quantum computing! The key observation is that actually fully factoring numbers is overkill. All we need is to do *something* that classical computers can't, and perhaps that something can be based on the hardness of factoring, without needing to return the factors themselves.

Intuition for how this might work can be found in the cryptographic concept of the *zero-knowledge proof*, which allows a prover to demonstrate that they know a particular fact or value to the verifier, without actually revealing any further information about it. As an example, consider the following (classical) protocol by which a prover can demonstrate that they know the discrete logarithm of a value without revealing it. [38] Suppose y, g, and p are publicly known integers, such that $y < p$, p is a large prime, and g is a generator for the multiplicative group of integers modulo p. The prover wants to demonstrate that they secretly hold a value x such that $g^x \bmod p \equiv y$. They first choose a random integer r such that $0 \leq r < p - 1$, compute $C = g^r \bmod p$, and send C to the verifier. The verifier now randomly chooses to ask for either the value r or the value $x + r$. Upon receipt, either can be easily checked: with knowledge of r, the verifier checks that C indeed is equal to $g^r \bmod p$; with $x + r$, the verifier checks that $g^{r+x} \equiv Cy \pmod{p}$. We observe two guarantees: first, the verifier gets no extra information from their receipt of *only* r or $x + r$—in either case, the information about x is perfectly statistically hidden by the randomness in r. Second, if the prover can consistently answer correctly over many repetitions of the above protocol, the verifier should be convinced that the prover knows *both* r and $r + x$ each time, and thus x! Crucially, the prover did not know which question would be asked until *after* making the commitment C. It's also crucial that a new r is chosen for each repetition, otherwise the verifier could easily extract x. This protocol is known specifically as a zero-knowledge *interactive* proof, because of the requirement that several messages to be sent back and forth between the prover and verifier. It has a structure shared by many zero knowledge interactive proofs: first, the prover makes a *commitment*, and then the verifier makes a *query* chosen at random from a set. Knowledge of the correct response to a single query is not sufficient to recover anything about the hidden information, but knowledge of the correct responses to *all* of the queries is sufficient to recover the hidden information in full. By repeating the protocol many times, the verifier becomes confident that the prover knows the answer to all

the queries simultaneously, and therefore knows the secret.

There is a really nice way that such a structure can be applied to the quantum setting. In the commitment phase, the prover commits to the claim that they hold a particular *quantum state*. Then, the verifier's queries can correspond to different ways of *measuring* that quantum state. Intuition from quantum state tomography tells us that if "correct"[8] measurement results can be produced for arbitrary measurement bases, the state can be reconstructed. Thus by repeating the protocol many times to ensure the prover always responds correctly regardless of the measurement basis, the verifier can ensure that the prover does indeed hold (or at least have a description of) the state to which they commit. This intuition only goes so far, however, because we desire that the protocols remain convincing even in the adversarial setting, where the prover is not just noisy, but instead is actively trying to fool the verifier by producing false measurement results. In this setting it is necessary to show more than just that the prover's data could reasonably have come from the committed quantum state. We also must ensure that there are no shortcuts by which a classical cheater could produce results that seem to follow the same distribution!

The first protocol with this structure was introduced to the literature in 2018, by Brakerski et al. [39] (although the general structure just described was not explicitly presented by the authors in that work). The authors construct a protocol by which the prover can use a cryptographic construction called a *trapdoor claw-free function* (TCF) to commit to holding a superposition of the form $|x_0\rangle + |x_1\rangle$, where x_0 and x_1 are n-bit (classical) strings. They use a cryptographic problem called Learning with Errors (LWE) to construct a TCF with the necessary properties. Importantly, the protocol is set up such that the specific values x_0 and x_1 are computationally hard for a classical cheater to find (under the LWE assumption). But, the classical verifier has access to some secret information, with which the two values are easy to compute given the prover's commitment. After the verifier has received the prover's commitment, they move onto the "query" phase, in which the verifier asks the prover to make one of two simple measurements: either measure all of the qubits in the computational (Z) basis, collapsing the superposition and yielding x_0 or x_1, or measure them all in the Hadamard (X) basis, yielding a measurement result that depends on quantum *interference* between x_0 and x_1. The authors show that a prover that is able to answer *both* queries consistently would be able to use those answers to break the LWE assumption, thereby bounding the probability by which a classical cheater could pass the protocol and providing a proof of quantumness.

Unfortunately, setting the cryptographic parameters to values that make classical cheating hard causes the quantum circuits to be so large that its implementation is quite far out of reach of near-term devices.[9] For this reason, further studies explored whether cryptographic problems other than LWE could be used in the same, or a similar, protocol. The main challenge is that the protocol requires a very strong cryptographic assumption called the

[8]Following the probability distribution of the committed state.

[9]No extensive analysis of the quantum resources needed for this protocol seems to have been published in the academic literature; however, a brief investigation done by myself together with Dr. Andru Gheorghiu (Chalmers University of Technology, Sweden) convinced me that it will not be feasible for some time.

adaptive hardcore bit assumption, which roughly states that given one of the two bitstrings in the superposition (say, x_0 above), it is computationally hard to find even a single *bit* of information about the second bitstring (x_1). To my knowledge, it is not known how to build a TCF with such a strong cryptographic guarantee from anything other than LWE.[10] Instead, studies have focused on modifying the protocol itself to relax the required cryptographic assumptions of the TCF.

The first paper to do so constructed a related protocol in the *random oracle model*, where both the prover and verifier have access to an oracle implementing a random function (the outputs of the function are perfectly random, but *consistent* when given the same input). [41] Intuitively, the randomness is used to "scramble" the values x_0 and x_1 before measurement, removing the extra structure a classical cheater could leverage (which created the need for the adaptive hardcore bit requirement). The challenge of using this protocol in practice, of course, is that random oracles do not exist in real life; they can be approximately implemented via the *random oracle heuristic* which replaces the random oracle with a cryptographic hash function.[11] Additionally, if the random oracle heuristic is applied, the protocol requires that the cryptographic hash function be evaluated coherently on a superposition of inputs, adding to the quantum circuit size. However, if one is willing to accept that, this protocol yields multiple benefits. First of all, as alluded to earlier, it reduces the cryptographic requirements of the TCF, removing the need for the adaptive hardcore bit property. Taking advantage of this fact, the authors provide a new TCF construction based on a computational problem called *Ring-LWE*, which is expected to be more efficient to implement than regular LWE. Additionally, the inclusion of the random oracle allows the protocol to use only a single round of messages between the prover and verifier—thus making it *non-interactive*, which could be useful in certain practical scenarios.

In Chapter 5 of this dissertation, we construct a protocol which removes the need for the adaptive hardcore bit property in the *standard model* of cryptography—that is, without the need for random oracles. This both makes a stronger demonstration of a fundamental difference in quantum versus classical computational power, and also removes the need for evaluating a cryptographic hash function coherently in addition to the TCF. In that chapter we also introduce two new TCF constructions, whose hardness are based on factoring and the decisional Diffie-Hellman (DDH) problems, respectively. The factoring-based construction requires the quantum prover to compute only $f(x) = x^2 \bmod N$ coherently, as opposed to the $f(x) = a^x \bmod N$ required by Shor's algorithm, leading to a dramatic reduction in the cost of implementation—yet the hardness of classically cheating remains the same. With the efficient circuit constructions that we describe in Chapter 7, we believe that this protocol is the closest yet to being implemented on real quantum devices in the next few years. As a

[10] There is one other proposal, based on isogeny-based group actions, that is reasonably *conjectured* to have the adaptive hardcore bit property. [40] In any case, that construction does not yield any benefits over LWE in terms of practical efficiency.

[11] The validity of the random oracle heuristic is a subject which has been explored at length; [42], [43] the broad consensus in the cryptographic community seems to be that despite the fact that it does not have any theoretical backing, it is fine in practice.

first step towards this goal, in Chapter 6 we present a first proof-of-concept experiment, in which the protocol is implemented at a small size in a trapped ion quantum computer.

Before concluding this section, I would like to describe two more papers which have taken new approaches to demonstrating quantum computational advantage. Neither seems to have any hope of being implemented on near term devices, but both make new progress in showing what *types* of cryptography can be used to build these protocols—and hopefully, they can lead to new constructions that are indeed cheaper to implement. The first is a protocol by Yamakawa and Zhandry, which operates in the random oracle model but requires no TCF at all—the random oracle is the only cryptographic tool required! [44] It introduces a clever construction called "quantum state multiplication", in which two quantum states $\sum_x c_x |x\rangle$ and $\sum_y d_y |y\rangle$ are combined into a single state $\sum_{x,y} c_x d_y |x+y\rangle$ (an operation that is usually impossible, but is made possible by the specific setup in the protocol). The second is a protocol by Kalai et al., which constructs a compiler that takes *multi-party* interactive proofs (that is, those with multiple provers working together to try to convince the classical verifier of a fact) and turns them into *single-prover* protocols. [45]

1.2.2.3 Moving past proofs of quantumness

While it seems to be an important milestone in the path towards full-scale quantum computing, demonstrating quantum computational advantage will not forever remain a particularly *useful* task. Fortunately, the protocols discussed in the preceding section represent just a subset of quantum interactive (or sometime non-interactive) protocols, which in general can achieve much more than simply demonstrating quantum computational power. In fact, the first protocol described above, based on the LWE problem, already pursued further applications. Not only was it presented as a test of quantumness, but it also represents a way to use an untrusted quantum device to generate random numbers that are *certifiably* quantum—that is, true randomness! In a pair of related papers, Mahadev showed that similar constructions could be used to implement classical homomorphic encryption for quantum circuits [46] and even the classical verification of *arbitrary* quantum computations [47]. Later papers also showed that this type of protocol could be used for other tasks, such as verifiable remote state preparation. [48] Finally, it has also been shown (in a paper I co-authored with several colleagues) that some of the quantum advantage protocols described in the previous section can be used *unchanged* to prove certain facts about the inner workings of a quantum device, leading to implications such as certifiable quantum random number generation directly from those protocols. [49]

Part I

Numerical studies of many-body quantum systems

In this first Part, we explore the following question: despite the seemingly fundamental fact that generic many-body quantum mechanical systems are exponentially costly to classically simulate, *how far can we push it*? While, as described in the Introduction (Chapter 1), classical simulation can actually be efficient for quantum systems with certain special properties like low entanglement, here we focus on instances in which this is *not* the case, and the best approach is truly to store and manipulate exponentially-large vectors of coefficients to represent quantum states. We apply the tools of modern supercomputing, including massive parallelism both via the distributed memory paradigm and also via acceleration on graphics processing units (GPUs), as well as cutting-edge algorithms for numerical linear algebra, to push our computations to as large system sizes as possible, enabling the numerical observation of new physics which was previously obscured by finite size effects.

In Chapter 2, we present the numerical package `dynamite`, which provides an easy-to-use and straightforward Python interface to highly-optimized, massively parallelizable numerics for many-body systems of quantum spin-1/2 particles with arbitrary interactions. In Chapter 3, we present a new numerical approach to the specific problem of exploring many-body localization in the Heisenberg model with disordered on-site fields. In particular, we apply a cutting-edge numerical algorithm called LOBPCG in combination with other algorithmic tricks and a hand-tuned software implementation to dramatically reduce the computer memory required to solve the problem.

Chapter 2

`dynamite`: massively parallel numerics for many-body quantum spin systems

2.1 Introduction

The spin-1/2 particle is one of the simplest and most famous quantum systems, and its dynamics and behavior were established early in the study of theoretical quantum mechanics. The Schrodinger equation for a single two-level system has a straightforward form, and in many cases can be solved analytically without much difficulty. However, the study of collections of such simple particles has proved much more difficult, and in many cases much more interesting. Emergent properties including novel phases of matter, some of which only arise in exotic circumstances such as out-of-equilibrium driven systems, have generated excitement and driven an intense push towards the deeper understanding of these systems' behavior. Throughout most of the 20^{th} century, advances in their study came largely from the development of increasingly clever analytic methods. But recently, concurrent, drastic improvements in both high-performance computing power and experimental control of individual quantum particles have enabled new forays into questions about many-body quantum mechanics which had remained stubbornly resistant to analysis.

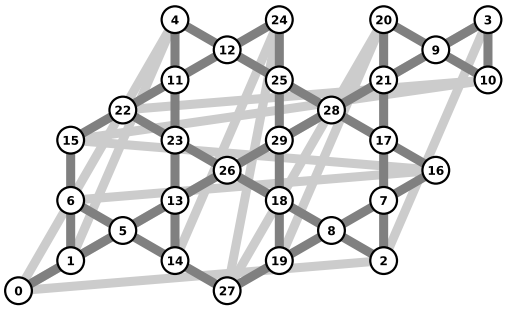
Even with the power of modern computers, the study of many-body quantum systems remains challenging. For example, simply representing an arbitrary quantum operator with support on L spin-1/2 particles into computer memory requires storing roughly 4^L complex numbers—a prospect which is infeasible even on a supercomputer, for L of just a couple dozen. Despite this, great progress has been made through the design of numerical algorithms which take advantage of the fact that the space of physically relevant quantum operators and states is often much smaller than the space of all possible ones. A classic and widely-used example of such a technique is the tensor network formalism, and more specifically matrix product states, which can be used to efficiently store and manipulate quantum states of low entanglement. A number of numerical libraries have been developed to perform tensor network computations, and they have become crucial tools in the field of numerical many-

```
from dynamite.operators import sigmax, sigmay, sigmaz

H = 0
for i, j in kagome_bonds:  # definition of kagome_bonds not shown
    # Heisenberg interaction between spins i and j
    H += sum(0.25*s(i)*s(j) for s in [sigmax, sigmay, sigmaz])

ground_state_energy = H.eigsolve()
```

(a) Code snippet solving for the ground state energy of the Heisenberg model on the Kagome lattice.



(b) A Kagome lattice geometry, for 30 spins on a torus. Light gray bonds represent "wrap-around" bonds due to the periodic boundary conditions.

N. spins	Dim. of H	Solve time
24	1,352,078	0:00:01.9
27	20,058,300	0:00:32.9
30	77,558,760	0:03:40.3

(c) Time to solve for the ground state, for various system sizes, on one Nvidia A100 GPU. Conservation laws were used to reduce the dimension of the Hilbert space, and the solve was performed in "matrix-free" mode.[1]

Figure 2.1: **An example application of `dynamite`: solving for the ground state of the Heisenberg model on the Kagome lattice.** A full script for this example, including code to compute the lattice bonds, is distributed with the `dynamite` source code, in the `examples/scripts/kagome/` directory.

body physics [50], [51]. However, quantum systems with large amounts of entanglement are not treatable with matrix product states—a set that includes some of the most exciting and mysterious phenomena, such as the many-body localization phase transition and the dynamics of quantum information scrambling. The numerical study of these systems requires storing the entire wavefunction explicitly; as a result, interrogating the phenomena that emerge only when L is large requires leveraging all of the tools of modern supercomputing, both in algorithms and hardware.

In this chapter, we present `dynamite`, a library for the numerical study of systems of spin-1/2 particles. Its main functions are to compute the time evolution of pure states in closed quantum systems, and to compute the eigenvalues and eigenvectors of quantum operators.

[1] In all cases, Hilbert space size is reduced by working in the half-filling subspace (this model conserves total magnetization). For $N = 24$ and $N = 27$, Hilbert space is reduced by another factor of 2 by applying a $\mathbb{Z}_2$ (spin-flip) symmetry. `dynamite` was built with real number scalars for these timing results.

This is achieved via *Krylov subspace algorithms*, in which a low-dimensional subspace that is continually updated throughout the computation until covergence is achieved—without relying on particular features of the wavefunction like low entanglement. Built on top of the PETSc and SLEPc sparse linear algebra libraries, `dynamite` is designed from the ground up to leverage the massive parallelism available in modern high-performance computing environments, with the ability to harness large numbers of CPUs distributed across many compute nodes via MPI, and to perform highly optimized computations on the thousands of cores of datacenter GPUs via CUDA. Previously, many-body physics studies looking to achieve parallelism at this scale required custom implementations in low-level languages like C, with much optimization required to achieve good performance. `dynamite` abstracts away this optimization from the user, offering a clean, intuitive Python interface in which quantum operators and states can be built symbolically (Figure 2.1a). This symbolic representation is then passed to a highly optimized backend, where features like GPU acceleration and memory saving "matrix-free" methods (see Section 2.4) can be enabled with the push of a button, enabling computations that are simultaneously very fast and memory-efficient (Figure 2.1c). At the same time, `dynamite` is portable, running equally as well in a Jupyter notebook on a laptop as it does in a massively distributed compute job on a cluster. This, along with its simple interface, makes it also useful as a tool for quick exploratory computations in the lab. Both operators and states can easily be exported from their symbolic representation to `numpy` format as well, allowing easy interface with existing code bases.

This chapter is organized as follows. In Section 2.2, we summarize the core functionality of `dynamite`, and also note how it is distinguished from several other software packages. In Section 2.3, we walk through several examples of studies that can be performed by `dynamite`, presenting along the way some novel physics results that `dynamite` was used to discover. In Section 2.4, we describe the algorithms and computational techniques underlying `dynamite`. In Section 2.5 we provide performance results for its solvers, measuring both elapsed time and memory usage in a number of different computational scenarios. Finally, in Section 2.6 we conclude with some discussion and outlook for the future.

2.2 Core functionality

`dynamite` offers the following core features:

- **Symbolic operators on arbitrary geometries** Symbolically build operators on collections of spin-1/2 particles with arbitrary connectivity, by summing n-body products of single-site Pauli operators. When computations are performed, `dynamite` automatically converts this symbolic representation in Python into the numerical one appropriate for the selected backend.

- **Time evolution** Given a time-independent Hamiltonian H, time t and a pure state $|\psi_0\rangle$, compute $|\psi_t\rangle = \exp(-iHt)\,|\psi_0\rangle$. Or, compute the imaginary time evolution $|\psi_\tau\rangle =$

$\exp(-H\tau)\,|\psi_0\rangle$. Piecewise-constant time-dependent Hamiltonians (such as Floquet systems) can also be used.

- **Eigensolving** Compute an operator's k lowest eigenvalues and their associated eigenstates, for user-configurable k. Or, additionally given a target energy τ, compute the k eigenvalues closest to τ and their associated states.

- **Entropy** Tracing out arbitrary sets of spins from Efficiently compute reduced density matrices from pure states, and the associated von Neumann or Renyi entropies.

- **Symmetry subspaces** Reduce computation time and memory usage by specifying subspaces corresponding to symmetries conserved by the Hamiltonian. `dynamite` has optimized implementations of $U(1)$ (conservation of total spin) and $\mathbb{Z}_2$ (spin flip) symmetries. It also allows users to define their own subspaces consisting of arbitrary sets of product states (for example, the allowed states in a system with Rydberg blockades).

- **Matrix-free methods** Drastically reduce memory usage by running computations "matrix free," where matrix elements are computed on the fly from the symbolic representation rather than being stored in memory. See Section 2.4.2 for details.

- **Extensive documentation, tutorials, and examples** Full documentation of all of `dynamite`'s functionality can be found at `https://dynamite.readthedocs.io/`. That site also includes instructions for installing the package and running the tutorial Jupyter notebooks, as well as a link to a large set of example scripts.

Some related software packages, whose functionality intersects with `dynamite`'s in various ways, include: QuSpin [52], a python package for exact diagonalization and dynamics that uses OpenMP parallelization; SPINPACK, a C package for finding lowest-energy eigenstates that uses MPI (or hybrid MPI-Pthreads) parallelization [53]; and SpinED, a static executable for calculating lowest-energy eigenstates (single node support) [54]. To our knowledge `dynamite` is the first numerical package that allows users to symbolically build quantum operators as Python objects, and then run computations on them in a massively parallel, distributed memory or GPU accelerated setting, including features such as matrix-free computation.

2.3 Examples

In this section we present a series of examples representing a range of interesting physics problems that can be studied with `dynamite`. Each of the following examples has a corresponding entry in the `examples/scripts` directory of the `dynamite` source tree, where users can find the full example source code and accompanying documentation. Those full examples also include various extra performance optimizations and other tricks, so the authors recommend that readers interested in reproducing or modifying these examples use those

scripts as a starting point. Two of the examples also correspond to new scientific insight that was gained through studies using `dynamite`; we discuss the results where applicable.

2.3.1 Floquet prethermalization

Closed quantum systems subjected to a drive tend to heat up to infinite temperature, because the drive inputs energy to the system but no energy is ever dissipated. At first sight, this means that most driven closed quantum systems are not particularly interesting—the infinite temperature state corresponds with a density matrix that is the identity, and is thus featureless. One way of avoiding this infinite-temperature fate is by setting up the system to exhibit *many-body localization* (MBL), which prevents the system from thermalizing, even at infinite time (MBL is explored further in Section 2.3.2). However, another way to observe interesting physics is to look at the system's behavior *pre-thermal* regime: the period of time before it has had the chance to fully heat up. Amazingly, analytic studies have shown that in certain Floquet systems—those in which the drive is periodic—this pre-thermal regime can exist for a time scale at least exponentially long in the frequency of the drive. [55]–[59] The hope is that such a long prethermal regime provides a sufficient window of time to observe interesting phenomena. Here, we numerically study the time evolution of a set of related Hamiltonians, and observe that long-range order induced by an effective Hamiltonian in the prethermal regime can lead to a novel out-of-equilibrium phase of matter called a *discrete time crystal*—in which a discrete time translation symmetry is spontaneously broken.

We begin by considering a 1D spin chain with open boundary conditions, with a long- or short-range ZZ interaction (either decaying as a power law, or nearest-neighbor). In addition we apply a nearest-neighbor XX interaction and a uniform, static magnetic field $\vec{h}$. This interaction has the form:

$$H = J_z \sum_{i<j} \frac{\sigma_i^z \sigma_j^z}{|i-j|^\alpha} + J_x \sum_{\langle i,j \rangle} \sigma_i^x \sigma_j^x + \sum_i \vec{h} \cdot \vec{\sigma} \tag{2.1}$$

where the angle brackets on the second term indicates it is only a nearest-neighbor interaction. We may consider the nearest-neighbor interaction to correspond to the case of $\alpha = \infty$. In `dynamite`, this Hamiltonian itself can be implemented in just a few lines of code, as shown in Code Listing 2.1.

For the Floquet drive, after every period T of time evolution the system will undergo a global π-pulse which we denote as X, rotating all spins by 180° around the x-axis. (Note that this is equivalent to flipping the direction of the magnetic field $\vec{h}$ across the x axis every time T).

There are a few quantities we would like to track during the evolution of this system. First, we'd like to measure the energy of the effective Hamiltonian D^* under which the system evolves (in a frame that flips along with the π-pulses X). The exact form of $D*$ cannot be expressed concisely, but we can approximate it as the Hamiltonian of Eq. 2.1 averaged over

```
def build_hamiltonian(alpha, Jz, Jx, h):
    # sums over all ranges of interaction
    # index_sum takes the interaction sigmaz(0)*sigmaz(r) and
    # translates it along the spin chain
    long_range_ZZ = op_sum(
        1/r**alpha * index_sum(0.25*sigmaz(0)*sigmaz(r))
        for r in range(1, config.L)
    )

    # an XX interaction on every neighboring pair of sites
    # the 0.25 is because spin operators are 1/2 times the Pauli
    nearest_neighbor_XX = index_sum(0.25*sigmax(0)*sigmax(1))

    # op_sum combines the three components of the magnetic field vector;
    # index_sum translates it to every site along the spin chain
    magnetic_field = index_sum(
        op_sum(hi*0.5*s() for hi, s in zip(h, [sigmax, sigmay, sigmaz]))
    )

    return Jz*long_range_ZZ + Jx*nearest_neighbor_XX + magnetic_field
```

Code Listing 2.1: Implementation of the long-range Hamiltonian in Eq. 2.1.

a full cycle of length $2T$, which we denote as D:

$$D = \frac{1}{2}\left(H + X^\dagger H X\right) \tag{2.2}$$

In addition to the expectation value of D, we compute the magnetization of each spin in the Z direction as the expectation values of S_i^z for all i, as well as the entanglement entropy. With that, we can implement the Floquet evolution of our system, as shown in Code Listing 2.2. Note how D and the global pi-pulse are extremely easy to implement in `dynamite` via its support for arbitrary operator arithmetic. Furthermore, both operators are very fast to compute because the arithmetic is implemented *symbolically* via Pauli operations, rather than matrix-matrix products.

Using `dynamite` to evolve this system yields the results plotted in Figure 2.2. Here, we have used a system size of $L = 22$ spins, with the parameters $\alpha \in \{1.13, \infty\}$ (for long- and short-range interactions respectively), $J_z = 1$, $J_x = 0.75$, and $\vec{h} = (0.21, 0.17, 0.13)$.[2] The frequency ω is the inverse of the time T that is allowed to elapse before the spin flip operator X is applied. In all cases the evolution starts from a product state; the precise product

[2]For the time evolution in the figure we have built the Hamiltonian with closed rather than open boundary conditions, for which we define the long range-interaction strength by replacing $|i-j|^{-\alpha}$ with $[(L/\pi)\sin|i-j|\pi/L]^{-\alpha}$, which matches the periodicity of the boundary conditions.

```
# the Hamiltonian; see previous code listing
H = build_hamiltonian(args.alpha, 1, args.Jx, args.h_vec)

# pi pulse operator
X = index_product(sigmax())

# the averaged "effective" Hamiltonian
D = (H + X*H*X)/2

# we create Deff and the Sz operators before the iterations start,
# in order to cache their matrix form
Sz_ops = [0.5*sigmaz(i) for i in range(args.L)]

# a workspace vector to store the output of the evolution in
tmp = state.copy()

for cycle in range(args.n_cycles):
    H.evolve(state, result=tmp, t=args.T)  # evolve under H for a time T
    X.dot(tmp, result=state)               # apply the pi pulse

    # "state" now contains the state vector at this time step
    # now compute some stats...

    half_chain_entropy = entanglement_entropy(state, range(args.L//2))

    D_val = D.expectation(state)

    Sz_vals = []
    for i in range(args.L):
        Sz_vals.append(Sz_ops[i].expectation(state))

    # now perhaps print the results, or save them to a file, or...
```

Code Listing 2.2: Time evolution under the Floquet Hamiltonian.

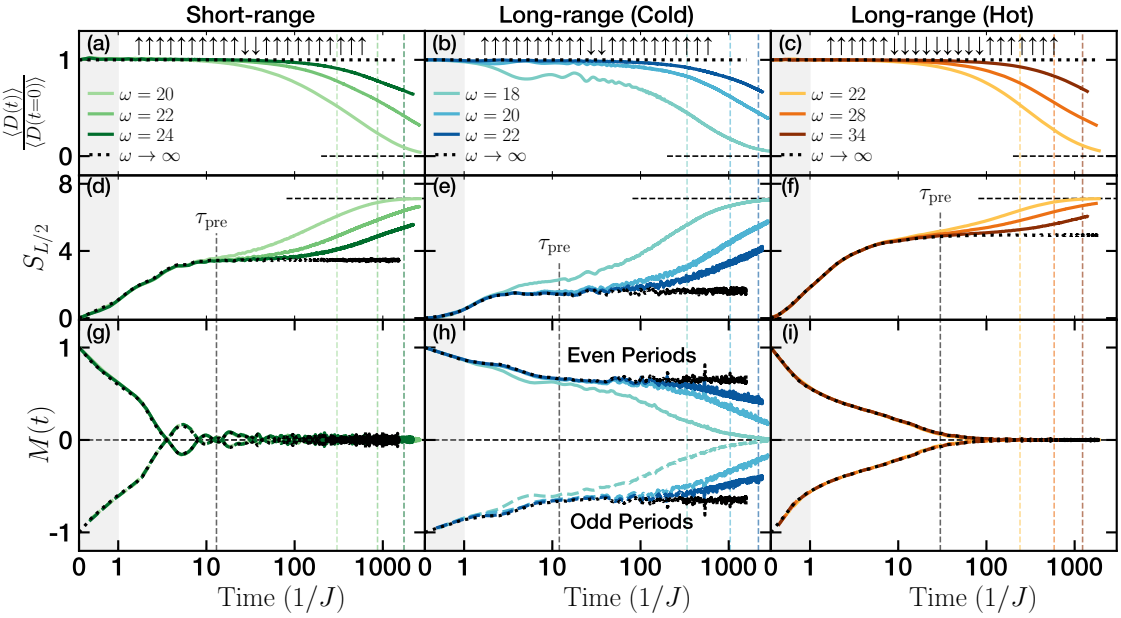


Figure 2.2: **Time evolution in a Floquet driven system, exhibiting prethermal discrete time crystalline behavior.** [60] The behavior in the limit $\omega \to \infty$ is computed by evolving directly under D, which is equal to D^* in this limit. (a-c) Expectation value of the energy with respect to D, the approximate effective Hamiltonian in the prethermal regime. The energy is approximately conserved during the prethermal regime, until the system heats to infinite temperature at a time scale exponentially long in ω. (d-f) Half-chain entanglement entropy. We observe that by the time τ_{pre}, the system thermalizes with respect to the effective pre-thermal Hamilonian, at which time the entropy hits a plateau; only when the system leaves the prethermal regime does the entropy approach its infinite temperature value. (g-i) Total magnetization. Period-doubling behavior, corresponding to spontaneous breaking of the discrete time-translation symmetry, is observed; it is stable only in the cold long-range case where D^* supports ferromagnetic order.

state is labeled in Figure 2.2 and is varied in the long-range case to explore both low-energy ("cold") and high-energy ("hot") initial states.

Begin by observing the top row of plots of Figure 2.2, which show the expectation value of the approximate effective Hamiltonian D throughout the time evolution. In every case (short-range interactions, and both temperatures of long-range interactions) we observe a clear signature of a pre-thermal regime corresponding to the effective Hamiltonian approximated by D. The expectation value of D is conserved for the period in which the system's evolution is well-approximated by it, and only after a sufficiently long time does the system begin to heat up and eventually reach a thermal state in which $\langle D \rangle = 0$. As expected, we observe that the length of the prethermal regime depends exponentially on the frequency ω. The same pattern is shown in the entanglement entropy, in the middle row of plots, with the

addition of the fact that we see the initial effective thermalization of the product state to the thermal state with respect to D! In particular, the initial product state has no entanglement, and after a time τ_{pre} (which does not depend on ω) the system reaches an entanglement entropy corresponding to a thermal state *of the same temperature as the initial state* (with respect to D). This is made particularly clear by comparing to the black dotted line, which plots the limit of $\omega = \infty$ (implemented by just evolving under D directly, which is exactly equal to D^* in this limit). The "kink" corresponding to where the system has thermalized with respect to D but has not yet left the prethermal regime is called the "prethermal plateau." Finally, in the bottom row of plots, we observe the time-crystalline behavior that was alluded to at the beginning of this section, in the form of *period-doubling*: the magnetization only returns to its original value every other period of the drive, corresponding to a spontaneously broken time translation symmetry. Note in particular that in the short-range and hot long-range cases, this time crystalline phase is not stable: it disappears by the time τ_{pre} is reached, because in these cases the effective Hamiltonian D^* is not capable of supporting long-range magnetic order (the time crystal "melts"). On the other hand, in the cold long-range case, D^* supports stable long-range ferromagnetic order, and the time crystalline behavior survives all the way until the system leaves the prethermal regime and begins to thermalize to its final infinite temperature state—a length of time that is exponentially dependent on the frequency. For more details and corresponding analytic analysis, see the published manuscripts corresponding to this study. [60], [61]

2.3.2 Many-body localization in the Heisenberg random field model

One of the main motivations for pushing numerics to the largest possible system sizes is to study emergent phenomena which simply cannot exist in just a few spins. A prototypical example is *many-body localization* (MBL): a surprising phenomenon in which certain many-body systems with sufficiently strong disorder fail to thermalize. In particular we will study the MBL transition, in which the system moves from thermalizing to localized as the disorder strength is increased. Characterization of this transition has remained elusive: finite size effects seem hard to avoid, and tensor network methods break down due to extensive entanglement in states near the transition. Thus, iterative methods like the Krylov subspace methods used by `dynamite` have proved to be one of the best tools for its study. [62], [63] We refer readers interested in learning more about the physics of MBL to one of the excellent review papers on the subject. [64]

Here, we show how to implement a model of nearest-neighbor Heisenberg interactions on a 1D chain with open boundary conditions, with disorder implemented as random Z fields on each site:

$$H = \sum_{\langle i,j \rangle} \vec{S}_i \cdot \vec{S}_j + \sum_i h_i S_i^z \tag{2.3}$$

where $\vec{S} = (S^x, S^y, S^z)$, the subscripts indicate the index of the spin in the chain, and the

angle brackets indicate that the indices run over nearest neighbors. The values of h_i are drawn from a uniform distribution $[-h, h]$ where h is a parameter that controls the strength of the disorder. The implementation in `dynamite` is presented in Code Listing 2.3. This Hamiltonian conserves total magnetization $\sum S^z$, which we use to reduce the size of the Hilbert space for our computations.

```
from dynamite.operators import sigmax, sigmay, sigmaz, op_sum, index_sum
from numpy import random

def build_hamiltonian(L, h):
    # Heisenberg interaction on all nearest neighbors
    H = index_sum(
        op_sum(0.25*s(0)*s(1) for s in [sigmax, sigmay, sigmaz]),
        size=L
    )

    # random fields
    H += op_sum(
        random.uniform(-h, h)*0.5*sigmaz(i)
        for i in range(L)
    )
```

Code Listing 2.3: **Implementation of the nearest-neighbor Heisenberg model, plus on-site random fields.**

In this example we identify the MBL transition via the *half-chain entanglement entropy* of eigenstates, $S_{L/2}$, which is simply the bipartite von Neumann entropy when half the spins are traced out. The MBL transition should correspond to a transition from volume law to area law entanglement. In the full example included with the `dynamite` source, we also examine an eigenvalue statistic called the "adjacent gap ratio."

The key feature that makes MBL so interesting is that the transition from volume to area law entanglement does not only occur in the ground state, but in excited states as well. This presents a great use case for `dynamite`'s "target" eigenvalue solver, which finds the k eigenvalues (and eigenvectors) closest to a target energy, where k is user configurable. Code Listing 2.4 implements the following computation: 1) choose an energy in the middle of the spectrum, 2) solve for 32 eigenpairs near this point, and then 3) compute the entanglement entropy for all of those eigenpairs.

An extensive numerical study of the MBL problem is presented in Chapter 3.

2.3.3 Many-body chaos in the Sachdev-Ye-Kitaev model

In spirit the Sachdev-Ye-Kitaev (SYK) model represents the opposite of the localization in the previous example: it is expected to scramble information highly efficiently via chaotic

```
from dynamite.subspaces import SpinConserve
from dynamite.computations import entanglement_entropy

# ...
# not shown: set L and h
# ...

H = build_hamiltonian(L, h)
H.subspace = SpinConserve(L, L//2)

# compute 32 eigenpairs near energy 0.1,
# which is in the middle of the spectrum
evals, evecs = H.eigsolve(target=0.1, nev=32, get_vecs=True)

# compute the half-chain entanglement entropy of each
entropies = []
for evec in evecs:
    entropies.append(
        entanglement_entropy(evec, range(L//2))
    )
```

Code Listing 2.4: **Implementation of the Heisenberg + random field model.** `build_hamiltonian` is implemented in Code Listing 2.3.

many-body dynamics. [65], [66] Indeed, it exhibits *maximal chaos*: the Lyapunov exponent, which characterizes the rate of chaos, saturates its upper bound of $2\pi T$, where T is the temperature of the system. [67] Its physics can be connected to the dynamics of quantum information in black holes, providing a testbed for exotic phenomena such as scrambling-based teleportation. [68]–[71]

The SYK model gives us a chance to look at how quantum systems other than spins can be explored with `dynamite`, by transforming them onto a spin system. The SYK model we'll use consists of Majoranas interacting in 0D, with random couplings. Specifically it consists of every possible 4-body interaction among N Majoranas, with each term having a random coupling strength:

$$H = \sqrt{\frac{6}{N^3}} \sum_{ijkl} J_{ijkl} \chi_i \chi_j \chi_k \chi_l \tag{2.4}$$

where J_{ijkl} are randomly chosen from a Gaussian distribution with variance 1.

To map the Majoranas onto the spin systems that are natively supported in `dynamite`, we can use the following transformation. For the Majorana with index i, let $q = \lfloor i/2 \rfloor$. Then

$$\chi_i = \sigma_q^{\{x,y\}} \prod_{m \in [0, q-1]} \sigma_m^z \tag{2.5}$$

where the first Pauli is σ^x if i is even and σ^y if it's odd. In words, the Majorana consists of a σ^x or σ^y with a string of σ^z extending to the edge of the spin chain. Note that we get two Majoranas for each spin! This is straightforward to implement in `dynamite`, but is actually already built in in the `dynamite.extras` module so we don't have to do it ourselves. Using that, an implementation of this Hamiltonian is shown in Code Listing 2.5. Note that this Hamiltonian has enough terms that building it can take an appreciable amount of time; a more efficient but less transparent version of the `build_hamiltonian` function can be found in the full SYK example, distributed with the source code of `dynamite`. The full example also takes advantage of a $\mathbb{Z}_2$ symmetry in the Hamiltonian to speed up the computations.

```
from dynamite.extras import majorana
import numpy as np

def build_hamiltonian(N):
    # N is the number of Majoranas

    H = 0
    for i in range(N):
        for j in range(i+1, N):
            for k in range(j+1, N):
                for l in range(k+1, N):
                    Jijkl = np.random.normal()
                    H += Jijkl*majorana(i)*majorana(j)*majorana(k)*majorana(l)

    return np.sqrt(6/N**3) * H
```

Code Listing 2.5: **Implementation of the SYK Hamiltonian (Eq. 2.4).**

We can study the fast scrambling behavior of the SYK model by examining *out of time order correlators* (OTOCs). In particular, we will measure to what extent two local operators $V(0)$ and $W(t)$ anticommute for various times t, where the anticommutator at time $t = 0$ is zero:

$$C(t) = \langle |\{W(t), V(0)\}|^2 \rangle. \tag{2.6}$$

It is helpful to reduce this to the following equivalent expression

$$C(t) = 2\mathrm{Re}\left[\langle W(t)V(0)W(t)V(0)\rangle\right] + 1/2 \tag{2.7}$$

which is the formulation of $C(t)$ that we will use here.

Defining $O(t) = W(t)V(0)W(t)V(0)$, we are specifically interested in this operator's expectation value with respect to thermal states, of various (inverse) temperatures β. Now, `dynamite`'s speed comes from the fact that it works with pure states, rather than mixed states—so the obvious plan to just compute $\mathrm{Tr}\left[O(t)e^{-\beta H}\right]$ is out of the question. Instead,

we can take advantage of an idea called "quantum typicality" to get an estimate of the expectation value more efficiently. [72], [73] Quantum typicality says that $\mathrm{Tr}\left[O(t)e^{-\beta H}\right]$ is well approximated by the expectation value of $e^{-\beta H/2}O(t)e^{-\beta H/2}$ with respect to random states (where we have split the thermal operator in half across the trace to get a more symmetric result). For simplicity we can set $W = \chi_0$ and $V = \chi_1$. With that, for a uniformly random state $|\psi_r\rangle$ we will compute (writing things out in full):

$$\langle\psi_r| e^{-\beta H/2}e^{iHt}\chi_0 e^{-iHt}\chi_1 e^{iHt}\chi_0 e^{-iHt}\chi_1 e^{-\beta H/2} |\psi_r\rangle \tag{2.8}$$

In Code Listing 2.6 we demonstrate how the expectation value of this operator can be computed in `dynamite`.

In Figure 2.3, we compute $F(t)$ which is closely related to $C(t)$: it is the "regularized" OTOC in which the thermal density matrix is spread equally across the operator, see the supplementary information of the published manuscript for details [74]. Note that there are a lot of independent parameters: the time t, temperature β, and system size N can all be varied, and to achieve good statistical significance requires then averaging over both the disorder in H and the random states used in the quantum typicality computation. Figure 2.3(a) shows the dependence of $F(t)$ on t for various system sizes, at $\beta J = 10$. (The plotted value is $\tilde{F}(t) = F(t)/F(0)$). There are two main observations to be made from this figure. First, the scrambling behavior is clear: $F(t)$ decays toward zero with a certain time scale, determined at early times by the *Lyanpunov exponent* λ as $1 - \tilde{F}(t) \sim e^{\lambda t}/N$. Second, there is clear flow of λ with system size. A finite-size rescaling procedure can be used to extrapolate λ to the limit of infinite system size, at which point it can be plotted as a function of the temperature βJ (Figure 2.3(b)). The observed dependence of λ tracks closely with the result of a semiclassical solution (the Schwinger-Dyson equations, labeled "SD" in the figure), and at large β (low temperature), the theoretically predicted fast-scrambling limit of $\lambda \leq 2\pi/\beta$. For more details, including of the finite-size rescaling procedure and the prediction from Schwinger-Dyson equations, see the published manuscript presenting these results. [74]

The results summarized in Figure 2.3 and elucidated in [74] provided the first numerical evidence supporting the claim that the SYK model exhibits fast scrambling behavior at low temperature. Previous efforts at smaller system sizes were unable to observe this behavior due to finite size effects; this study was made possible by `dynamite`'s speed at very large system sizes which enabled analysis of Hamiltonians on up to 60 Majoranas. Due to the four-body all-to-all connectivity of the model (Eq. 2.4), the Hamiltonian has an extremely large number of terms in the SYK Hamiltonian (487,635 for a system size of 60 Majoranas!), explicitly storing the (sparse!) matrix in memory is infeasible for large problems: for 60 Majoranas it would require over 350 terabytes of RAM! Thus `dynamite`'s matrix-free ("shell") mode, which only stores and manipulates the operator symbolically, was crucial for this study.

After these results were published, two studies followed in which I was not directly involved, but warrant mention for the curious reader. The many-body teleportation of quantum states via scrambling, a phenomenon that emphasizes the duality between quantum mechanical models like SYK and the dynamics of gravity models, in particular black holes

```
from dynamite import config
from dynamite.extras import majorana
import numpy as np

# ... define number of majoranas N, temperature beta, time t, etc ...

# number of spins is half the number of majoranas that live on them
# this configures the number of spins globally
config.L = (N+1)//2

W = majorana(0)
V = majorana(1)

H = build_hamiltonian(N)

# imaginary time evolution for e^-beta*H/2
ket = H.evolve(State(state='random'), t=-1j*beta/2)
bra = ket.copy()

# need a temporary "workspace" vector as well
tmp = ket.copy()

for _ in range(2):  # need to apply this twice
    # V(0)
    V.dot(tmp, result=ket)

    # W(t)
    H.evolve(ket, t=t, result=tmp)
    W.dot(tmp, result=ket)
    H.evolve(ket, t=-t, result=tmp)

# finally take the inner product and get the result!
otoc_val = bra.dot(ket)
C_t = 2*otoc_val.real + 1/2
```

Code Listing 2.6: **Computation of the expectation value of the out-of-time-order correlator with respect to a thermal state, via quantum typicality.** Here we work through Eq. 2.8 from right to left, applying time evolution and operators as we go. Since the thermal imaginary time evolution is symmetric on both sides of the Equation, we "re-use" it rather than computing it twice.

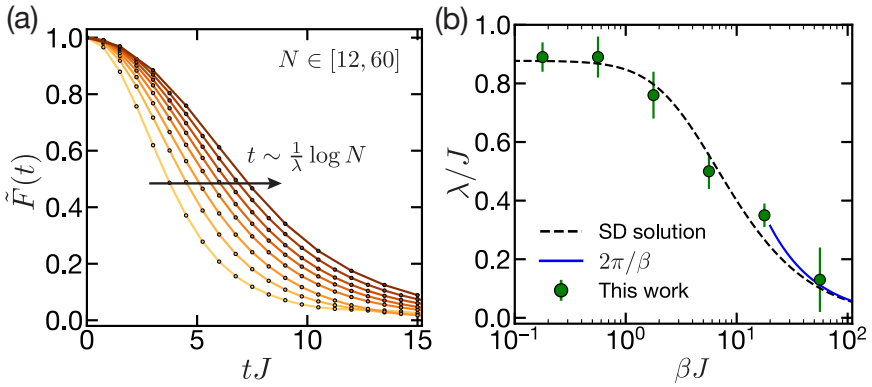


Figure 2.3: **Fast scrambling in the SYK model.** [74] (a) The value of the regularized and normalized OTOC $\tilde{F}(t) = F(t)/F(0)$ (essentially equivalent to $C(t)$) as a function of time t, for varying system sizes N. Clear scrambling behavior is observed in the exponential growth of $1 - \tilde{F}(t)$ at short times, from which a Lyapunov exponent λ can be extracted. (b) The result of extrapolating λ to infinite system size, for various values of the temperature β. The results agree with the semiclassical prediction from the Schwinger-Dyson equations, and at low temperature saturate the fast scrambling limit of $\lambda \leq 2\pi/\beta$.

and wormholes, was numerically observed with `dynamite`. [71] In another study, `dynamite` was used to show that *sparse* SYK, in which many of the J_{ijkl} are zero, can reproduce many of the features of the regular SYK model, but with a much lower computational cost to the simulation. [75]

2.4 Methods

In this section, we give an overview of the numerical techniques used by `dynamite`, including their strengths and limitations. First we describe at a high level the intuition behind Krylov subspace algorithms. Next we describe the symbolic representation of operators used internally by `dynamite`, and how it is used to implement the "matrix-free" multiplication of states by operators. Finally we describe techniques used for parallelization, with a focus on the matrix-free operations.

2.4.1 Krylov subspace algorithms

The two main computationally intensive tasks that `dynamite` performs are time evolution and eigensolving. At first glance, for an operator H, the computational task of quantum time evolution is to compute the unitary $\mathcal{U} = \exp(-iHt)$, and for eigensolving it is to compute a matrix A and diagonal matrix Λ such that $AH = \Lambda H$. Both $\mathcal{U}$ and (A, Λ) can be computed in a straightforward manner using standard linear algebra subroutines (for example, SciPy's `linalg.expm` and `linalg.eigh` routines); however, as we explain below,

doing so is exponentially more expensive than is necessary for most physics studies. For an quantum operator on a Hilbert space of dimension N, matrices $\mathcal{U}$ and A both of dimension $N \times N$, consisting of 4^L complex numbers for a generic system of L spins-1/2 particles. The power of the Krylov subspace algorithms underlying `dynamite` stems from the fact that they avoid ever explicitly computing $\mathcal{U}$ or A, only requiring $\mathcal{O}(2^L \cdot \text{poly}(L))$ memory to operate and having a corresponding improvement in runtime. Here we give a brief overview of how they work; for details please see the SLEPc Users Manual and Technical Reports. [76], [77] Note that if the `dynamite` user wants to tune the solver by passing any of the "command line" options to `SLEPc` as described in those documents, it can be achieved via the `slepc_args` argument to `dynamite.config.initialize()`.

An initial concern seems to be that the matrix H itself is of dimension $N \times N$, which ostensibly is an obstacle to achieving the efficiency just promised. However, H has a crucial feature that $\mathcal{U}$ and A do not: it is *sparse*. This means that most of the matrix elements are zero, and the cost of storing and working with the matrix can be reduced dramatically by only storing the non-zero elements. In fact, the memory usage can be reduced even further by not storing any of the matrix elements at all, as we discuss below in Section 2.4.2. With that, we move on to discussing how we can use this sparse H to achieve performant time evolution and eigensolving.

For time evolution, the key is that we are often more interested in the *action* of the matrix $\mathcal{U}$ on a state vector than we are in the matrix itself. For an initial state $|\psi_0\rangle$, we desire to directly compute the time evolved vector $|\psi_t\rangle = \exp(-iHt)\,|\psi_0\rangle$ without needing to compute the unitary as an intermediate step. The Taylor expansion of the matrix exponential suggests that for reasonably small $||Ht||$, $|\psi_t\rangle$ will be dominated by a linear combination of the vectors $H^i\,|\psi_0\rangle$ for $i \in [0, m]$, where m is a small integer, say 30. These vectors are easy to compute by repeatedly multiplication with H. This intuition forms the core idea behind Krylov subspace methods: construct the subspace $\mathcal{K}_m = \text{span}\left(\{|\psi_0\rangle, H\,|\psi_0\rangle, H^2\,|\psi_0\rangle, \cdots, H^{m-1}\,|\psi_0\rangle\}\right)$, and solve the linear algebra problem on this much smaller subspace. For time evolution, this means projecting H onto $\mathcal{K}_n$ to yield a matrix $\tilde{H}$ of dimension only $m \times m$, computing the matrix exponential $\tilde{\mathcal{U}} = \exp(-i\tilde{H}t)$ (a trivial computation for a small matrix like $\tilde{H}$), and then computing $|\psi_t\rangle \approx \tilde{\mathcal{U}}\,|\psi_0\rangle$. Projected back into the full Hilbert space, for small $||Ht||$, $\tilde{\mathcal{U}}$ should be a good approximation of $\mathcal{U}$, and thus the result should be a good approximation of $|\psi_t\rangle$.

The case in which $||Ht||$ is not sufficiently small presents the opportunity to discuss another powerful technique crucial to these algorithms, which is called *restarting*. Instead of computing a single Krylov subspace and attempting to perform the whole computation in it, the subspace is iteratively updated throughout the computation until convergence is achieved. The default restarting scheme for time evolution in `dynamite` is quite simple: the desired evolution of time t is broken down into a series of shorter evolutions of time Δt, which is sufficiently small that $\tilde{\mathcal{U}}$ is a good approximation of $\mathcal{U}$. After each evolution of Δt, the Krylov subspace is constructed anew, such that it tracks the evolution of the state vector through the Hilbert space. This is the algorithm used by the EXPOKIT package, which is reproduced in `SLEPc`; [76], [78] there is also a Krylov-Schur solver with a less intuitive

restarting scheme available in SLEPc but we find it to be less stable.

For eigensolving, we take advantage of the fact that computing the matrix A of all eigenvectors, which is enormous, may be overkill. Physics studies are often interested in only a few particular eigenstates, such as the ground state and first few excited states. Matrix-vector products are again useful here, with the simplest algorithm being the so-called "power method." Consider the decomposition of a random vector in the basis of (unknown) eigenstates of H: it will have the form $|\psi_r\rangle = \sum_i c_i |\phi_i\rangle$, where each $|\phi_i\rangle$ is an eigenstate. When H is multiplied by this vector several times, the result is $H^m |\psi_r\rangle = \sum_i c_i \lambda_i^m |\phi_i\rangle$—the eigenvectors with eigenvalues of largest absolute value have been enhanced exponentially; for sufficiently large m the result will converge to the most extremal eigenvector. Once again, this process can be enhanced through the use of a Krylov subspace: instead of discarding the vectors $|\psi_r\rangle$ through $H^{m-1} |\psi_r\rangle$, they are used to construct a Krylov subspace. The matrix H is projected into this subspace and the full spectrum is computed of the much smaller $\tilde{H}$. As in the case of time evolution, a single iteration of this algorithm is unlikely to converge within a reasonable error tolerance, again calling for a *restarting* scheme. A great deal of effort has gone into designing optimal restarting schemes for Krylov eigensolving; they are out of the scope of this work but we refer interested readers to the SLEPc technical papers for a discussion of the solvers underlying dynamite [77].

Perhaps surprisingly, since it solves for extremal eigenvalues, this technique can also be used to compute sets of neighboring excited states anywhere in the spectrum. This is achieved via a *spectral transform*: rather than performing the solve on the matrix H itself, it is performed on a function of H which moves the desired eigenvalues to the outside of the spectrum. In dynamite, the default spectral transform for finding interior eigenpairs is *shift-and-invert*, which, for a target energy σ, performs the transformation $H \to (H - \sigma)^{-1}$ such that the eigenvalues closest to σ lie on the outside of the transformed matrix. This technique has been used with great success in the study of many-body localization in particular, as discussed in Sec. 2.3.2.

Finally, we note an important feature of all of these techniques: they almost exclusively rely on the matrix-vector product as the definition of the matrix, rather than ever needing to access individual matrix elements themselves. This fact is crucial for the ability to use "matrix-free" implementations with these algorithms, as discussing in Sec. 2.4.2 below.

2.4.1.1 Error

Despite being frequently termed "exact diagonalization," Krylov subspace algorithms are approximate. In dynamite, the error is managed entirely by the SLEPc library that implements the solvers; SLEPc applies techniques such as the "restarting" just described in order to converge results to within a requested tolerance ϵ (which can be passed as an argument in dynamite). In this section, we make precise what dynamite's error tolerance means, largely by describing the definitions of error used by SLEPc.

For eigensolving, there is a straightforward way to compute the error. Suppose, for a matrix H, that $\tilde{\lambda}$ and $\tilde{x}$ are an approximate eigenvalue and eigenvector, respectively. Ideally,

we would have $A\tilde{x} = \tilde{\lambda}\tilde{x}$ exactly. To capture the essence of this goal, the *residual vector* is defined as

$$r = A\tilde{x} - \tilde{\lambda}\tilde{x} \tag{2.9}$$

and we may define the error in terms of this residual vector. For the solver settings used by `dynamite`, convergence to a tolerance ϵ is achieved when at least the requested number of eigenpairs have

$$||r||_2/|\lambda| < \epsilon \tag{2.10}$$

This can be shown to correspond with the following two error bounds on $\tilde{\lambda}$ and $\tilde{x}$:

$$|\lambda - \tilde{\lambda}| \leq \epsilon|\tilde{\lambda}| \tag{2.11}$$

$$\sin\theta(x, \tilde{x}) \leq \frac{||r||_2}{\delta} \tag{2.12}$$

where $\theta(x, \tilde{x})$ is the angle between the exact and approximate vectors and δ is the distance from $\tilde{\lambda}$ and its closest neighboring eigenvalue. Note in particular that the second expression implies that nearly-degenerate eigenvalues can lead to considerable mixing of the corresponding eigenvectors. Also note that when a spectral transform is used (the `target` mode of eigensolving) the convergence criterion is evaluated for the *transformed* eigenproblem. Finally, we observe that the Krylov-Schur method, used by `SLEPc` for eigensolving, is known to sometimes "miss" degenerate (or very near degenerate) eigenvalues; it should not be relied upon to always return the correct multiplicity for a degenerate eigenspace. [79] A future version of `dynamite` may explore switching to the LOBPCG solver (see Chap. 3) which does not suffer from this limitation. A detailed description of the convergence criteria for `SLEPc`'s eigensolvers, including flags that can be used to adjust these criteria, can be found in Section 2.5 of the `SLEPc` Users Manual. [76] (See the documentation of `dynamite`'s `config.initialize()` function for information on how to pass "command-line" flags to `SLEPc`).

For time evolution (action of the matrix exponential), unfortunately there is not such a straightforward way of computing the error. As before, for an approximate output vector $\tilde{w} \approx e^{-iHt}x$ we may define the residual vector

$$r = e^{-iHt}x - \tilde{w} \tag{2.13}$$

but in this case there is no way to explicitly compute it. Instead we must rely on indirect methods of estimating and controlling the error. `dynamite` uses `SLEPc`'s implementation of the EXPOKIT solver for the action of the matrix exponential; the solver uses internal error metrics to attempt to keep the magnitude of the residual vector smaller than the requested tolerance ϵ. The details of how this is done is out of the scope of this manuscript, but can be found in the original work detailing the EXPOKIT solver. [78] Note that there is no explicit guarantee that the error will remain below the tolerance, but the error is usually well-controlled in practice. Bounding the error of Krylov subspace techniques for quantum time evolution is a long-standing and ongoing area of research; improved error estimates could potentially be included in a future version of `dynamite`. [78], [80]–[86]

2.4.2 Matrix-free computation

Most operators that one encounters in the study of many-body spin physics are *sparse*: an overwhelming fraction of the matrix elements are zero. It is for this reason that `dynamite` uses the sparse numerical linear algebra libraries `PETSc` and `SLEPc` for its solvers. By default, the solvers use the *compressed sparse row* (CSR) storage format for matrices, in which only nonzero matrix elements (and the necessary indices) are stored in memory. For an operator of dimension $2^L \times 2^L$, this reduces the cost of storage (and matrix-vector multiplication) from $\mathcal{O}(4^L)$ to $\mathcal{O}(2^L k)$, where k is the number of nonzero elements per row—and k is polynomial in L for virtually all physically relevant operators. Of course, $\mathcal{O}(2^L k)$ still has the potential to be quite large, and frequently is in practice. For the case that even the sparse representation of an operator uses too much memory, `dynamite` implements a custom representation of operators which we call the "XZC" representation, that leverages the tensor product structure of spin chain operators to store them using as little as $\mathcal{O}(k)$ memory.[3] The intuition is that the operator is represented symbolically as a decomposition into strings of Pauli matrices; instead of being stored, the elements of the large sparse matrix are computed "on-the-fly" as they are needed. In the field of numerical linear algebra, this type of technique is known as "matrix-free" computing; for historical reasons it is referred to as "shell matrices" in `dynamite`. It dramatically reduces the memory usage, enabling computations that wouldn't be possible otherwise due to memory limitations—especially on GPUs, where VRAM is limited. Usually, this comes with a trade-off: the need to compute matrix elements on-the-fly adds to the computational cost, so the user pays for the reduction in memory usage with an increase in the runtime. However, CSR matrix operations are frequently limited by *memory bandwidth* rather than CPU usage—the speed is limited by how quickly the CPU can retrieve matrix elements from main memory. By reducing the amount of data transferred across the memory bus, it's actually possible for matrix-free methods to be *faster*, while simultaneously decreasing the memory usage! Whether this is the case in practice depends on how fast the matrix elements can be generated in real time—and much effort in the development of `dynamite` has been directed at optimizing this process to an extreme. In Section 2.5, we compare the performance of shell and non-shell computations for various problems. Here we describe how it works, and how it interfaces with the solvers.

The fundamental idea behind the XZC representation is that any operator on a set of L spins can be decomposed as a linear combination of products of single-site Pauli operators. Explicitly, for an operator on L spins, which has m terms in this decomposition, we may define a pair of $m \times L$ binary matrices $M^{(x)}$ and $M^{(z)}$, and a length-m complex-valued vector C, to represent an operator H as

$$H = \sum_i H_i = \sum_i C_i \bigotimes_j (\sigma_j^x)^{M_{ij}^{(x)}} (\sigma_j^z)^{M_{ij}^{(z)}} \tag{2.14}$$

where σ_j^x and σ_j^z are the Pauli X and Z operators on site j. In words, each row of $M^{(x)}$ and $M^{(z)}$, and corresponding element of C, corresponds to one term of the operator. In

[3] The XZC representation is currently denoted "MSC" in various parts of the `dynamite` source code.

each row, the 1's in each binary matrix represents the locations in which a σ^x or σ^z Pauli should be present (with 0's corresponding to the identity Pauli on that site). σ^y operators are represented via the product $\sigma_x\sigma_z$, when both $M^{(x)}_{ij}$ and $M^{(z)}_{ij}$ are equal to 1.

Because L in practice never exceeds 64, $M^{(x)}$ and $M^{(z)}$ are stored as vectors of integers.[4] The bits of each integer represent the spin indices for which a σ_x and/or σ_z operator should be included for that term. For the C_i, it is straightforward to see that each value must be either entirely real or entirely imaginary if the operator is Hermitian; this allows C to be stored as a vector of real values (64-bit floating point numbers). All together, an operator with m terms is stored as two length-m vectors of integers plus one length-m vector of real numbers, yielding an overall storage cost of $\mathcal{O}(m)$. The quantity k mentioned in the previous paragraph corresponds to the number of unique rows of $M^{(x)}$; in most situations m is proportional to k.

We now describe how computations are implemented on this format—specifically matrix-vector multiplication, which is by far the most important matrix operation for the iterative solvers described in Section 2.4.1. By default, `dynamite` uses product states in the Z basis as a basis for its representation of the Hilbert space. For the full Hilbert space of length 2^L, `dynamite` takes basis state of index α to correspond to the product state $|\alpha\rangle = \bigotimes_j |\alpha_j\rangle$, where α_j is the j^{th} bit of the binary representation of the integer α (and with $|0\rangle$ corresponding to an "up" spin, with σ_z eigenvalue $+1$, and $|1\rangle$ corresponding to a "down" spin, with eigenvalue -1). Then, an arbitrary state vector $|\psi\rangle$ is stored as an array $\vec{x}$ of 2^L complex numbers x_α, such that $|\psi\rangle = \sum_\alpha x_\alpha |\alpha\rangle$. This yields a very straightforward implementation of multiplication of operators by states. The operator-state multiplication can be decomposed as as the sum over multiplications by each of the H_i of Eq. 2.14:

$$H|\psi\rangle = \sum_{i,\alpha} H_i x_\alpha |\alpha\rangle \tag{2.15}$$

It is straightforward to see that the product of a single term H_i from Eq. 2.14 with a product state $|\alpha\rangle$ yields another product state $|\beta\rangle$ multiplied by a coefficient:

$$H_i|\alpha\rangle = C_i \bigotimes_j (\sigma^x_j)^{M^{(x)}_{ij}} (\sigma^z_j)^{M^{(z)}_{ij}} |\alpha_j\rangle = (-1)^{\sum_j M^{(z)}_{ij}\alpha_j} C_i |M^{(x)}_{i,*} \veebar \alpha\rangle \tag{2.16}$$

where $M^{(x)}_{i,*}$ is the i^{th} row of M. In words, the result product state is $|\beta\rangle$, where β is the bitwise XOR of the integer α and the i^{th} row of $M^{(x)}$, multiplied by the coefficient C_i, with a sign flip if an odd number of bits are set in the bitwise AND of the i^{th} row of $M^{(z)}$ and α.

This is very promising from an implementation perspective. As described above, the rows of $M^{(x)}$ and $M^{(z)}$, as well as the product states α, will be represented as integers, this operation can be performed extremely quickly using bitwise operations. Computing $\beta = M^{(x)}_{i,*} \veebar \alpha$ costs a single bitwise XOR. Computing the sign flip costs just a bitwise AND,

[4] `dynamite` uses 32-bit integers by default, but can be configured to use 64-bit integers if L is to exceed 31.

a `popcount` operation to count the number of bits set[5], and another bitwise AND to get the least significant bit of the result (the parity of the number of bits set). Finally flipping the sign of C_i costs a single XOR instruction, and then a fused multiply-add (one machine instruction in modern CPUs) can be used to multiply x_α by the sign-flipped C_i and add it into the appropriate element y_β of the output vector. Overall, computing the action of each term of H_i on each basis state costs two bitwise XORs, two bitwise ANDs, a popcount, and a fused multiply-add—not very expensive at all!

With that, we have a straightforward algorithm for computing a matrix-vector multiplication using the XZC representation of operators, given as pseudocode in Code Listing 2.7. The code listing includes one small addition that has not yet been discussed, which is the conversion of indices to states and vice versa. As previously mentioned, when using the full Hilbert space this operation corresponds to the identity—index α corresponds to product state $|\alpha\rangle$. However when the Hilbert space dimension is reduced into a symmetry subspace, this may not be the case. The functions `state_to_idx` and `idx_to_state` represent the conversion between indices and integers representing the corresponding product states.

2.5 Performance results

In this section we present performance results for `dynamite`, measuring both runtime and memory usage. We compare the performance of various computational settings, including single-core, multiple cores on the same compute node, multiple cores distributed across several compute nodes, and GPU computations. We also benchmark compressed sparse row (CSR) format for matrix storage versus `dynamite`'s custom matrix-free XZC format (see Section 2.4.2). We benchmark using three different Hamiltonians (introduced in the examples of Sec. 2.3) which are representative of the range of models `dynamite` is useful for. Importantly, they cover a wide variation in number of non-zero elements per row of the matrix, as described in Table 2.1—which is one of the primary factors influencing computational cost. The Heisenberg + random fields and SYK models also have conserved quantities that allow us to benchmark `dynamite`'s performance when working in subspaces.

All benchmarks in this section were performed on the Department of Energy's Perlmutter supercomputer at NERSC. The CPU benchmarks were performed on nodes having two AMD EPYC 7763 processors, each with 64 cores, for a total of 128 cores per node. Each node has a total of 512 Gb DDR4 RAM, with 256 Gb attached to each processor. All GPU benchmarks were performed using one Nvidia A100 GPU with up to 80 Gb attached VRAM, connected to one AMD EPYC 7763 processor (of which a single core was used), with 256 Gb DDR4 RAM. For more details on Perlmutter's specifications, see the cluster documentation. [87]

In Tables 2.2 and 2.3 we report performance results for the `eigsolve` and `evolve` methods of `dynamite` respectively. For eigensolving, we measure the cost of solving for the ground state of the model, to the default tolerance of 10^{-8}. For time evolution, we benchmark the

[5] `popcount` is a native machine instruction in most modern CPUs.

```
# matvec() computes y += H*x, where H is represented in XZC form by:
# - Mx (array of integers)
# - Mz (array of integers)
# - C  (array of real numbers)
function matvec(Mx, Mz, C, x, y)
    for alpha in 0:dim(x)
        for i in 0:nterms
            in_state = idx_to_state(alpha)
            out_state = Mx[i] ^ in_state  # bitwise XOR
            beta = state_to_idx(out_state)

            # & means bitwise AND
            flip_sign = parity_of_set_bits(Mz[i] & in_state)

            if flip_sign
                coeff = -C[i]
            else
                coeff = C[i]
            end

            y[beta] += coeff*x[alpha]
        end
    end
end
```

Code Listing 2.7: **Pseudocode for "matrix-free" matrix-vector multiplication, in the XZC representation.** This is the basic structure used by `dynamite`, although with many optimizations layered on top.

Model	Non-zero elements per row	Conserved quantities
Long-range (Eq. 2.1)	$2L$	(none)
Heisenberg + random fields (Eq. 2.3)	$\leq L$	Total magnetization
SYK (Eq. 2.4)	$L^4/24 + \mathcal{O}(L^3)$	Parity

Table 2.1: **The models used for benchmarking in Sec. 2.5.** The number of nonzeros per row for the Heisenberg model is variable because some of the off-diagonal terms cancel to zero; it is bounded by the given figure. Note that the long-range model, despite its all-to-all connectivity, has only a linear number of nonzero elements per row because the long-range ZZ terms all sum together on the diagonal.

evolution of a product state[6] to a time $t = 50/||H||_\infty$, again to within the default tolerance of 10^{-8}. We normalize t in this way because the computational cost of time evolution in `dynamite` scales with $||Ht||$; the normalization accounts for the arbitrary energy scale in the definition of each Hamiltonian and allows a fair comparison across models and system sizes. We note that the default tolerances are almost certainly tighter than is necessary for most physics studies; in practice users may want to reduce the tolerance to improve speed. In this section, all reported timings are real elapsed ("wall") time, and include all phases of the computation including setup steps like initialization of MPI and construction of the operators.

From the data in the two tables we can make a few observations. In single-core CPU performance, we observe that the CSR implementation outperforms the matrix-free implementation by a considerable amount. This makes sense, considering that the matrix-free implementation has to do more computational work to compute the matrix elements on-the-fly instead of pulling them from memory. The tradeoff is clear, however, in the memory usage, which is always less for the matrix-free implementation than CSR. Furthermore we observe that, as expected, the reduction in memory usage for the matrix-free implementation depends strongly on the number of nonzero elements per row of the model (see Table 2.1). SYK, which has the most nonzeros per rows, has the matrix-free implementation showing a dramatic reduction in memory usage of over 24× compared with CSR, for $L = 22$. This gap is expected to only increase at larger system sizes, but the memory usage for CSR matrices for SYK is so large for $L = 28$ spins that we are not able to collect performance data for that case: estimating the memory usage with `dynamite`'s `Operator.estimate_memory()` function suggests that it would require over 100 terabytes to store such a matrix!

The performance tradeoff picture changes, however, when we look at parallel performance—both in the case of multicore CPU and GPU computations. In several cases, the matrix-free implementation actually *outperforms* the CSR implementation! There is actually a very straightforward explanation for this behavior: the computation is not actually limited by a computational bottleneck, but by the *memory bandwidth*. Because the 128 cores on one node, and the many thousands of cores in the GPU, share memory, the memory bus is not sufficiently fast to keep all of them supplied with data—and in the CSR implementation, there is simply no way around this, because the matrix elements need to be pulled from memory. On the other hand, the matrix-free implementation puts significantly less pressure on the memory bus, and the addition of the extra cores gives enough computational power to handle the extra computation required to compute the matrix elements when needed. Essentially the only thing that needs to be drawn from memory in that case is the vector data itself—and even with that, `dynamite`'s matrix-free implementations are designed to make use of the on-chip memory cache as much as possible to reduce this. Furthermore, except perhaps in the case of large SYK Hamiltonians where there are a very large number of terms of the Hamiltonian, the XZC representation used to generate the matrix elements can *fully* remain in cache. Consistent with these facts is that the speed ratio between the CSR

[6]The performance of `dynamite`'s time evolution solver is not expected to depend on the initial state.

Model	Configuration			Eigensolve	
	L	Hardware	Matrix mode	Wall time	Memory [GB]
Heisenberg +random fields	24 dim: 2,704,156 nonzeros: 24	1 CPU core	CSR	0:00:12.808	1.618
			Matrix-free	0:01:05.621	0.867
		1 CPU node (128 cores)	CSR	0:00:04.648	11.714
			Matrix-free	0:00:03.965	11.248
		GPU	CSR	0:00:06.309	≤ 80
			Matrix-free	0:00:03.974	≤ 80
	30 dim: 155,117,520 nonzeros: 30	1 CPU node (128 cores)	CSR	0:02:37.938	141.844
			Matrix-free	0:05:26.727	59.624
		GPU	Matrix-free	0:02:00.793	≤ 80
Long range	22 dim: 4,194,304 nonzeros: 44	1 CPU core	CSR	0:04:07.932	5.069
			Matrix-free	0:06:22.741	1.311
		1 CPU node (128 cores)	CSR	0:00:17.864	19.154
			Matrix-free	0:00:19.227	11.347
		GPU	CSR	0:00:18.184	≤ 80
			Matrix-free	0:00:09.749	≤ 80
	28 dim: 268,435,456 nonzeros: 56	1 CPU node (128 cores)	Matrix-free	0:40:37.351	89.840
		GPU	Matrix-free	0:18:39.491	≤ 80
SYK	22 dim: 2,097,152 nonzeros: 7,547	1 CPU node (128 cores)	Matrix-free	0:26:44.837	17.131
		2 CPU nodes (256 cores)	CSR	0:11:31.164	472.640
		GPU	Matrix-free	0:28:45.358	≤ 80

Table 2.2: **Cost of solving for the ground state, for various models and configurations.** Exact memory usage not reported for GPU runs because tooling was not available to measure the peak GPU memory usage; bounds correspond to the amount of attached memory on the GPUs used for testing.

Model	Configuration			Time evolution	
	L	Hardware	Matrix mode	Wall time	Memory [GB]
Heisenberg +random fields	24 dim: 2,704,156 nonzeros: 24	1 CPU core	CSR	0:00:18.479	2.306
			Matrix-free	0:01:35.774	1.573
		1 CPU node (128 cores)	CSR	0:00:02.773	12.702
			Matrix-free	0:00:05.348	12.193
		GPU	CSR	0:00:06.887	≤ 80
			Matrix-free	0:00:04.113	≤ 80
	30 dim: 155,117,520 nonzeros: 30	1 CPU node (128 cores)	CSR	0:02:02.834	180.944
			Matrix-free	0:04:22.168	97.270
		GPU	Matrix-free	0:01:30.241	≤ 80
Long range	22 dim: 4,194,304 nonzeros: 44	1 CPU core	CSR	0:01:09.491	6.146
			Matrix-free	0:01:32.764	2.384
		1 CPU node (128 cores)	CSR	0:00:06.958	19.593
			Matrix-free	0:00:06.524	12.343
		GPU	CSR	0:00:19.050	≤ 80
			Matrix-free	0:00:04.386	≤ 80
	27 dim: 134,217,728 nonzeros: 54	1 CPU node (128 cores)	CSR	0:02:45.128	331.246
			Matrix-free	0:02:58.955	84.435
		GPU	Matrix-free	0:01:10.321	≤ 80
SYK	22 dim: 2,097,152 nonzeros: 7,547	1 CPU node (128 cores)	CSR	0:04:20.826	412.341
			Matrix-free	0:02:42.996	16.776
		2 CPU nodes (256 cores)	CSR	0:02:42.236	470.659
			Matrix-free	0:01:59.353	67.093
		GPU	Matrix-free	0:02:45.066	≤ 80

Table 2.3: **Cost of time evolution of a random state to a time $t = 50/||H||_\infty$, for various models and configurations.** Exact memory usage not reported for GPU runs because tooling was not available to measure the peak GPU memory usage; bounds correspond to the amount of attached memory on the GPUs used for testing.

implementation and the matrix-free one scales with the number of non-zero elements per row—corresponding to the fact that memory bandwidth is more of a bottleneck when there are more matrix elements. To summarize, in the parallel setting, it is often preferable to use the matrix-free implementation, because it simultaneously is faster and uses less memory.

Finally, we may compare the performance across the different hardware types. As is expected, in all cases using a single core is the slowest. In most cases, the matrix-free GPU performance is fastest, varying from a few times faster than the performance of 128 CPU cores to roughly comparable speed, depending on the model and problem size. However, we note that 128 CPU cores consist of an entire CPU node on Perlmutter, while there are 4 A100 GPUs per GPU node. So, for tasks such as averaging over disorder realizations, where computations can be "embarrassingly parallelized" by running one instance on each GPU on a node, the total computational throughput of a GPU node will in all cases exceed that of a CPU node by several times. (We also note that Perlmutter's 128 cores per CPU node is on the large end; many compute clusters have only 32, 48, or perhaps 64 cores per node). The conclusion is that, if GPUs are available, they are usually the fastest option. The only drawback to running on them is that there is hard limit to the system sizes they can run due to the limit of their total attached memory. On the Perlmutter GPUs, which have at most 80 GB of attached memory, even in matrix-free mode computations are limited to $L = 27$ spins with no subspaces in use, and perhaps a few more with the use of subspaces (e.g. $L = 30$ for the Heisenberg model with total magnetization conservation enabled), due to the memory cost just of storing state vectors. Parallelizing a single computation across several GPUs in `dynamite` is currently in development, but is not yet supported. Thus for reaching the very largest system sizes, the only option is to use CPU parallelism, perhaps across several nodes if necessary.

2.6 Discussion and outlook

In numerical studies of quantum many-body phenomena, one cannot avoid the information-theoretic fact that the number of classical bits required to store an arbitrary quantum state is exponential in the number of particles of the quantum system. In certain cases, such as those involving low amounts of entanglement, there are ways to approximate a subset of states efficiently, but the generic case does not permit such optimizations. Fortunately, with the tools of modern supercomputing it is possible to perform linear algebra on astoundingly large vector spaces, with dimension exceeding even one trillion—making it possible to directly numerically study many-body quantum systems of nontrivial size.

In this chapter we have presented `dynamite`, a software package for the numerical study of many-body systems of spin-1/2 particles (and other systems which can be mapped onto them). `dynamite` provides a straightforward and intuitive Python interface, yet the computations are performed by a highly optimized backend built from the ground up to enable massive parallelization on modern supercomputer clusters, whether via distributed memory CPU computing or acceleration on GPUs. Particularly notable is `dynamite`'s symbolic rep-

resentation of quantum operators, by which computations can be performed "matrix-free," reducing memory usage considerably while often also improving speed.

Moving forward, the most obvious next step is to implement multi-GPU computations in `dynamite`, which will help alleviate the main limitation of GPU computing in `dynamite`—the limited memory attached to a single GPU. This feature is already in development; multi-GPU computing only recently has become a serious tool in supercomputing and thus its support on the supercomputers to which we have access is imperfect, but we expect multi-GPU `dynamite` to be available in production once these issues have been worked out. We also note that large-memory GPUs are currently in high demand for machine learning applications as well, so improvements in the VRAM per GPU can be expected on the hardware side as well. Aside from that, a crucial feature of `dynamite` is its ability to take advantage of symmetries in the physical system to reduce the dimension of the Hilbert space on which computations will be performed. So, another promising path forward is to implement more of these conservation laws, perhaps including conservation of momentum corresponding to lattice symmetries. Finally, there are more improvements to be had on the algorithmic side. For example, leveraging recently discovered algorithms and new types of hardware has been shown to have the potential to accelerate certain computations dramatically [88].

In Chapter 3, we discuss how a relatively new algorithm called LOBPCG can be pushed to improve the performance of the mid-spectrum eigensolving that we explored in Sec. 2.3.2, with the help of a hand-tuned matrix-vector multiplication designed specifically for the Heisenberg model.

Chapter 3

A Scalable Matrix-Free Iterative Eigensolver for Studying Many-Body Localization

3.1 Introduction

A fundamental assumption in the traditional theory of statistical mechanics is that an isolated system will in general reach an equilibrium state, or *thermalize*. As early as the mid-20$^{\text{th}}$ century, Anderson demonstrated that a single particle moving in a highly disordered landscape can violate this assumption [89]. While surprising, that result does not readily extend to many-particle systems that exhibit strong interactions between the constitutent particles. The question of whether a similar effect could manifest in a strongly-interacting many-body system remained open for decades. This elusive phenomenon has been termed "many-body localization" (MBL).

Recently, advances in both high performance computing and experimental control of individual quantum particles have begun to yield insight into MBL. Both experimental [90]–[95] and numerical [63], [96]–[99] results have shown evidence of localization in small strongly-interacting multiparticle systems of 10-20 spins. Unfortunately, extrapolating results from these small system sizes to the infinitely-large thermodynamic limit has proven difficult. This lack of clarity has inspired a vigorous debate in the community about precisely what can be learned from small-size results. For example, it has been proposed that certain features do not actually exist at infinite system size [100], and even that MBL itself is only a finite-size effect [101]!

The primary goal of most studies is to identify and characterize a *localization transition*. In the thermodynamic limit, as the strength of the system's disorder increases, theory predicts a sharp, sudden change from a thermal to a localized state. Unfortunately, in the small systems available for study, that sharp transition turns into a smooth *crossover*, leading to the confusion about what constitutes the transition itself. Numerical evidence suggests that

the transition sharpens rapidly as system size increases, so accessing as large systems as possible is imperative for investigating MBL.

In pursuit of that goal, Luitz et al. used large-scale numerical linear algebra to show a localization transition for system sizes up to $L = 22$ [63], and in a following paper extracted useful data up to $L = 24$ [62]. In order to compute interior eigenstates for the MBL problem, the shift-and-invert Lanczos algorithm was used in combination with sparse direct solvers for solving the linear systems. One of the major disadvantages of this technique is that constructing the LU factorizations becomes extremely memory demanding, due to the so called fill in, for large number of spins L. Table 3.1 shows that the memory footprint of the LU factorization computed via STRUMPACK [102] grows rapidly as function of L. See also [62]. Hence, thousands of nodes on modern high performance computing infrastructures are needed to go beyond $L = 24$.

L	n	STRUMPACK	LOBPCG(64)
16	12,870	66 MB	8 MB
18	48,620	691 MB	31 MB
20	184,756	8 GB	118 MB
22	705,432	92 GB	451 MB
24	2,704,156	1 TB	2 GB
26	10,400,600	15 TB	7 GB

Table 3.1: **Comparison of memory footprint**. Total memory footprint is presented as a function of the spin chain length L for LU factorizations, computed via STRUMPACK, and the new matrix-free LOBPCG algorithm, with block size 64. The problem size is given by n.

In this paper, we introduce an new approach based on the locally optimal block preconditioned conjugate gradient (LOBPCG) algorithm to overcome the memory bottleneck that the shift-and-invert Lanczos algorithm faces. As shown in Table 3.1, we are able to reduce the memory footprint by several orders of magnitude, e.g., from 15 TB to only 7 GB for $L = 26$. This new approach will enable us to simulate spin chains on a single node, even up to $L = 24$. For larger spin chains we only require a few nodes.

The paper is organized as follows. We first review the Heisenberg spin model and MBL metrics in Section 3.2. Next, the LOBPCG eigensolver with efficient matrix-free block matrix-vector operations is discussed in Section 3.3. Then in Section 3.4, we illustrate the new matrix-free LOBPCG eigensolver for Heisenberg spin chains of sizes up to $L = 26$. Finally, the main conclusions are formulated in Section 3.5.

3.2 Problem Formulation

In this section we briefly review the properties of the spin chain model that most frequently is studied by numerical simulations of MBL.

3.2.1 Heisenberg Spin Model

We consider the nearest-neighbor interacting Heisenberg spin model with random on-site fields:

$$H = \sum_{\langle i,j \rangle} \vec{S}_i \cdot \vec{S}_j + \sum_i h_i S_i^z, \tag{3.1}$$

where the angle brackets denote nearest-neighbor i and j, h_i is sampled from a uniform distribution $[-w, w]$ with $w \in \mathbb{R}_0^+$, and

$$\vec{S}_i \cdot \vec{S}_j = S_i^x S_j^x + S_i^y S_j^y + S_i^z S_j^z,$$

where $S_i^\alpha = \frac{1}{2}\sigma_i^\alpha$, with σ_i^α the Pauli matrices operating on lattice site i and $\alpha \in \{x, y, z\}$. The parameter w is called the *disorder strength*, and is responsible for inducing the MBL transition. The values h_i are sampled randomly each time the Hamiltonian is instantiated, and the relevant physics lies in the statistical behavior of the set of all such Hamiltonians. The individual Hamiltonians H with independently sampled h_i are called *disorder realizations*.

Note that in (3.1) each term of each sum has an implied tensor product with the identity on all the sites not explicitly written. Consequently, the Hamiltonian for L spins is a symmetric matrix of dimension $N = 2^L$ and exhibits the following tensor product structure

$$\begin{aligned} H = &\sum_{i=1}^{L-1} I \otimes \cdots \otimes I \otimes H_{i,i+1} \otimes I \otimes \cdots \otimes I \\ &+ \sum_{i=1}^{L} I \otimes \cdots \otimes I \otimes h_i S_i^z \otimes I \otimes \cdots \otimes I, \end{aligned}$$

where $H_{i,i+1} = S_i^x S_{i+1}^x + S_i^y S_{i+1}^y + S_i^z S_{i+1}^z$ is a 4-by-4 real matrix and I is the 2-by-2 identity matrix. Remark that by definition, all matrices $H_{i,i+1}$ are the same and independent of the site i. For our experiments, we use open boundary conditions, meaning that the nearest-neighbor terms do not wrap around at the end of the spin chain. Open boundary conditions can be considered to yield a larger *effective* system size because of the reduced connectivity.

The state of each spin is described by a vector in $\mathbb{C}^2$, and the configuration of the entire L-spin system can be described by a vector on the tensor product space $(\mathbb{C}^2)^{\otimes L}$. In this specific case, however, the Hamiltonian's matrix elements happen to all be real, so we do not include an imaginary part in any of our computations. Furthermore, our Hamiltonian commutes with the total magnetization in the z direction, $S^z = \sum_{i=1}^L S_i^z$. Thus it can be block-diagonalized in sectors characterized by $S^z \in [-\frac{L}{2}, -\frac{L}{2}+1, \ldots, \frac{L}{2}-1, \frac{L}{2}]$. The vector

space corresponding to each sector has dimension $n = \binom{L}{S^z + \frac{L}{2}}$ such that the largest sector's dimension is $n = \frac{L!}{(\frac{L}{2})!(\frac{L}{2})!}$, and this corresponds to the actual dimension of the matrices on which we operate, see Table 3.1. While these subspaces are smaller than the full space, their size still grows exponentially with the number of spins L. Thus, the problem becomes difficult rapidly as L increases. Furthermore, the density of eigenvalues in the middle of the spectrum increases exponentially with L. Thus the tolerance used to solve for these internal eigenvalues must be made tighter rapidly as L increases.

3.2.2 Metrics for Localization

With the problem's matrix clearly defined, we now need a way of quantifying localization from the eigenvalues and eigenvectors. There are multiple quantities that can be used for this purpose. We focus on two here: one based on the eigenvalues, and one on the eigenvectors. The eigenvalue-based method (adjacent gap ratio) has been used in multiple previous works [96], [97], [101], [103], but suffers from large statistical noise and thus requires many samples to be usable. To reduce the number of samples required, we focus on the eigenvector-based method in our experiments.

3.2.2.1 Adjacent Gap Ratio

Random matrix theory informs us that the statistical distribution of eigenvalues will differ between localizing and thermalizing Hamiltonians [103]. In particular, we expect eigenvalues of a thermal Hamiltonian to *repel* each other, i.e., hybridization of eigenvectors prevents them from generally coming too close to one another. The eigenvalues of a localized Hamiltonian should not display this behavior: we expect them to be Poisson distributed. Therefore, we can measure localization by comparing the relative size of gaps between the eigenvalues. Thermal Hamiltonians will generally have more consistently sized gaps due to level repulsion.

The *adjacent gap ratio* is defined as follows

$$r_i = \frac{\min(\Delta_i, \Delta_{i+1})}{\max(\Delta_i, \Delta_{i+1})}, \qquad \Delta_i = \lambda_i - \lambda_{i-1},$$

where the eigenvalues λ_i are sorted in increasing order. Quantitatively, random matrix theory can inform the precise values we expect in the two cases, averaged over many pairs of neighboring eigenvalues. In the thermal case, we expect $\langle r \rangle \sim 0.53$, while for localizing Hamiltonians we expect $\langle r \rangle \sim 0.39$ [103].

3.2.2.2 Eigenstate Entanglement Entropy

The eigenvectors of the Hamiltonian can also help inform us about localization. In a thermal system, we expect quantum entanglement to be widespread, while in a localized system, the entanglement is not expected to be extensive. This idea can be quantified by choosing a *cut* which divides the spin chain into two pieces, and measuring the entanglement across it. In

practice, this entanglement is measured by removing one of the two pieces (by computing a partial trace), and then measuring the increase in entropy due to its removal.

Mathematically, for an eigenvector x, the entanglement entropy between two subsystems A and B can be computed as follows. Define $\rho \equiv xx^\top$ as the *density matrix* corresponding to the state x, represented as a column vector. Now let $\rho_A \equiv \mathrm{Tr}_B[\rho]$ be the density matrix of subsystem A, where Tr_B is the partial trace over sites in subsystem B. The entanglement entropy is then

$$S_{AB} = -\mathrm{Tr}\left[\rho_A \ln \rho_A\right].$$

Numerically, this quantity is generally computed in the following way: (i) compute ρ_A directly from x, (ii) compute the eigenvalues λ_i of ρ_A, and (iii) compute $S_{AB} = -\sum_i \lambda_i \log \lambda_i$. Note that in the first step, we do not hold ρ itself at any point since it is a dense matrix of dimension $n \times n$, and thus is not feasible to store in memory. Fortunately, it is not hard to compute the partial trace directly from x itself.

In this paper, we focus on the case in which we cut exactly in the middle of the spin chain, such that subsystems A and B are the left and right halves of the system. In this case, for eigenvectors corresponding to eigenvalues near 0, we expect the thermal case to have entanglement entropy [104]

$$S_{L/2} = \frac{L \ln 2}{2} - 0.5 \tag{3.2}$$

In the localized case the entanglement entropy will not scale with L, but instead will attain some constant value. For finite system sizes, we simply expect the entanglement entropy to decrease from the above value as the system becomes more localized.

Not only the value of the entropy changes during the localization transition: the statistics change as well. When compared across disorder realizations, the thermal entanglement entropy should consistently be the value in Equation (3.2), and thus have small variance. During the transition, however, we expect the entanglement entropy to depend strongly on the specific disorder realization and thus the statistic will have a large variance. Empirically, examining the *variance* of the entanglement entropy is one of the best ways to identify the localization transition.

3.2.3 Multiple Levels of Concurrency

The MBL study allows for at least 4 levels of concurrency. The first level corresponds to the need of averaging over (many) different and independently sampled *disorder realizations* in order to obtain relevant statistical behavior. Since the *disorder strength* is responsible for inducing the MBL transition, we also have to vary the disorder strength, giving rise to the second level of concurrency. The third level corresponds to the *eigenvalue chunks*, i.e., for each (large) eigenvalue problem, originating from one disorder realization and a particular disorder strength, we have to compute eigenvalues from different regions of the spectrum and their corresponding eigenvectors.

All previous levels of concurrency are completely independent and can be implemented in a massively parallel fashion by making use of iterative eigensolvers. The next level of parallelism takes place within these eigensolvers. Although most iterative eigensolvers follow a rather sequential procedure, each of the different steps within one iteration can be implemented in parallel.

3.3 Matrix-Free LOBPCG Eigensolver

The Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) algorithm [105], [106] is a widely used eigensolver for computing the smallest or largest eigenvalues and corresponding eigenvectors of large-scale symmetric matrices. Key features of the LOBPCG algorithm are: (i) It is matrix-free, i.e., the solver does not require storing the coefficient matrix explicitly. It access the matrix by only evaluating matrix-vector products; (ii) It is a block method, which allows for efficient matrix-matrix operations on modern computing architectures; (iii) It can take advantage of preconditioning, in contrast to, for example, the Lanczos algorithm.

In Section 3.3.1 we review the LOBPCG algorithm. Next, we discuss in Section 3.3.2 how the LOBPCG algorithm can be modified in order to compute interior eigenvalues and its corresponding eigenvectors. Finally, we explain in Section 3.3.3 how the (block) matrix-vector products can be efficiently implemented in parallel, both in OpenMP and MPI.

3.3.1 LOBPCG Eigensolver

Let $H \in \mathbb{R}^{n\times n}$ be a symmetric matrix and denote its eigenvalues and corresponding eigenvectors by $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ and $x_1, x_2, \ldots, x_n$, respectively. Then the diagonal matrix of the first $k \leq n$ eigenvalues $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_k)$ and the rectangular tall-skiny matrix of corresponding eigenvectors $X = [x_1, x_2, \ldots, x_k]$ satisfy the following eigenvalue problem

$$HX = X\Lambda$$

and X is the solution to the trace minimization problem

$$\min_{X^\top X = I} \mathrm{trace}\left(X^\top H X\right). \tag{3.3}$$

A similar trace maximization property exists for the eigenvectors corresponding to the k largest eigenvalues of H.

The basic idea of the LOBPCG method introduced by Knyazev [105] is to solve this trace optimization problem only locally in every iteration, in order to converge to the smallest (or largest) eigenvalues and corresponding eigenvectors. This yields the following updating formula

$$X_{i+1} = \mathrm{arg}_{X\in\mathcal{Z}} \min_{X^\top X = I} \mathrm{trace}\left(X^\top H X\right),$$

where

$$\mathcal{Z} = \text{span}\left\{W_i, X_i, X_{i-1}\right\},$$

with X_i and X_{i-1} the current and previous iterates, respectively, and W_i the preconditioned residual

$$W_i = K^{-1}\left(HX_i - X_i\Theta_i\right),$$

with K any preconditioner and $\Theta_i = X_i^\top H X_i \in \mathbb{R}^{k\times k}$. Note that W_i corresponds to the preconditioned gradient of the Lagrangian

$$\mathcal{L}(X,\Lambda) = \frac{1}{2}\text{trace}\left(X^\top H X\right) - \frac{1}{2}\text{trace}\left(X^\top X\Lambda - \Lambda\right)$$

associated to (3.3) and evaluated at (X_i, Θ_i).

The approximations to the smallest k eigenvalues and eigenvectors, so called Ritz pairs $(\tilde{\lambda}_j, \tilde{x}_j)$, can numerically be obtained from the Rayleigh–Ritz method, i.e.,

$$\Theta_i V = V\tilde{\Lambda},$$

where $\tilde{\Lambda} = \text{diag}(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_k)$ is a diagonal matrix containing the Ritz values on its diagonal and $V = [v_1, v_2, \ldots, v_k]$. The corresponding Ritz vectors are given by $\tilde{x}_j = X_i v_j$, for $j = 1, 2, \ldots, k$.

A basic version of the LOBPCG method, given in algorithm 3.1, is relatively easy to implement, however, it can suffer from numerical instability if not implemented carefully. Therefore we used the robust variant, introduced in [106], in all our experiments.

3.3.2 Computing Interior Eigenvalues

The LOBPCG algorithm has several advantages, such as blocking and preconditioning, compared to the Lanczos algorithm. However, the standard LOBPCG algorithm, as well as the standard Lanczos algorithm, only allow for computing the lower or upper part of the spectrum.

Within the Lanczos algorithm this issue is solved by a *shift-and-invert* transformation

$$(H - \sigma I)^{-1},$$

where $\sigma \in \mathbb{R}$ is the shift. This spectral transformation maps the eigenvalues closest to the shift σ to the outer part of the transformed spectrum which then can be efficiently computed by the shift-and-invert Lanczos algorithm. The big downside of this transformation is that it requires a memory demanding LU factorization for inverting the shifted matrix. This makes it impractical for large numbers of spins. As reported in [62], the computational cost of the overall algorithm is dominated by the construction of the LU factorization.

In order to avoid storing the matrix and computing memory demanding LU factorizations, we will make use of a different spectral transformation, the so called *spectral fold* [107]

$$(H - \sigma I)^2,$$

Algorithm 3.1: Basic LOBPCG algorithm

Input: number of eigenpairs k and block size $b \geq k$
$X_0 \in \mathbb{R}^{n \times b}$: matrix of starting vectors with $X_0^\top X_0 = I$
Output: $X \in \mathbb{R}^{n \times k}$: matrix of approximate eigenvectors
$\Lambda \in \mathbb{R}^{k \times k}$: diagonal matrix of approx. eigenvalues

1 Residual: $R = HX_0 - X_0(X_0^\top H X_0)$.
2 Initialize: $X = X_0$, $P = []$, and $n_c = 0$.
while $n_c < k$ **do**
3 Apply preconditioner: $W = K^{-1}R$.
4 Subspace: $Z = [W, X, P]$.
5 Rayleigh–Ritz:

$$(Z^\top H Z) \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} \Lambda$$

6 Update: $X \leftarrow WV_1 + XV_2 + PV_3$ and $P \leftarrow WV_1 + PV_3$.
7 Residual: $R = HX - X\Lambda$.
8 Update number of converged eigenpairs n_c.
end
9 Return first k columns of X and leading $k \times k$ block of Λ.

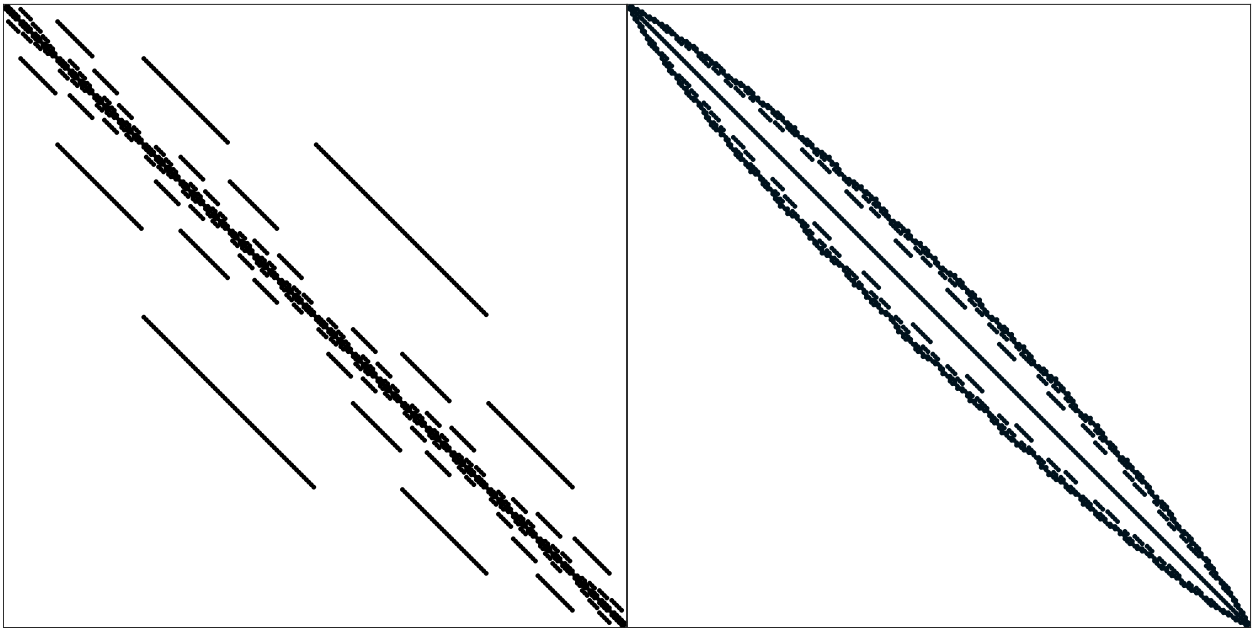
where $\sigma \in \mathbb{R}$ is again the shift. This transformation maps all eigenvalues to the positive real axis and the ones closest to the shift σ to the lower edge close to 0. Hence, we can use the LOBPCG eigensolver in combination with matrix-free block matrix-vector operations in order to compute interior eigenvalues and their corresponding eigenvectors. Because the transformed eigenvalue problem

$$(H - \sigma I)^2 x = \lambda x$$

is symmetric positive definite, we will use a diagonal (Jacobi) preconditioned conjugate gradient (PCG) method as preconditioner for the LOBPCG eigensolver, so that we again can make use of matrix-free block matrix-vector operations.

3.3.3 Matrix-Free Matrix-Vector Product

In contrast to the shift-and-invert Lanczos algorithm, where the dominant computational cost is the construction of the LU factorization, the dominant computational cost of the LOBPCG algorithm is the (block) matrix-vector product. Note also that by applying a spectral fold transformation, the matrix-vector product of the transformed matrix can be implemented simply by repeatedly applying a standard matrix-vector product of our Hamiltonian.



(a) OpenMP state ordering (b) MPI state ordering

Figure 3.1: **Sparsity structure of the Hamiltonian.** Here, $L = 10$. (a) the OpenMP state ordering and (b) the MPI state ordering.

In order to implement this matrix-vector (MATVEC) operation matrix-free and efficiently, we must consider our choice of basis states, as well as our ordering of these states in the vector. We use two different orderings, one for the pure OpenMP and the other for the hybrid MPI–OpenMP implementation. They are chosen to optimize for SIMD vectorization and communication bandwidth respectively. To conserve memory, we compute all off-diagonal matrix elements on the fly, avoiding explicitly storing them.

3.3.3.1 Basis States

A convenient basis for the Hamiltonian is the Z-polarized product state basis: the states in which every spin is in a Z-eigenstate. These states can be represented compactly as a bitstring, with zeros representing the spin-up state and ones representing the spin-down state (or vice versa). This representation, which is also used in [108], allows fast computation of the values of matrix elements via bitwise operations. For example, the value of the term $\sum_{\langle i,j \rangle} S_i^z S_j^z$ can be computed in just a couple of operations:

```
inline double ZZ(int state, int L) {
  // number of terms in the sum that are -0.25
  int n_negative = __builtin_popcount(state ^ (state>>1));
  return 0.25*(L - 2*n_negative - 1);
}
```

3.3.3.2 Data Layout of Block Vectors

LOBPCG is a block solver, meaning that it operates on a block of several vectors at the same time, usually 32 or 64 in our case. A crucial performance consideration is how this data should be stored. When viewing this block as a tall, skinny dense matrix, there are two obvious possibilities: row- or column-major. Other possibilities such as Z Morton ordering exist, but there is no clear reason that they would give performance gains in this context.

Both intuitively and empirically, we find that row-major ordering yields a faster matrix-vector multiplication than column-major does. From a data-locality perspective this makes sense [109]. When we are performing the multiplication for a particular matrix element, in the row-major case the sequence of relevant values in the input vector block are contiguous in memory, meaning that they can all be fetched in the same cache line. Furthermore, the multiplication can be vectorized with SIMD instructions. In the column-major case, those same vector values are spread out in memory by a distance equal to the vector length, and must be accessed separately. Furthermore, there is danger of concurrency issues with false sharing in the column-major case. In particular, if threads are each given a unique set of matrix rows to compute, they will never write to the exact same places in the output vector. However, in the column-major case two threads are more likely to write to *nearby* locations in memory. If these nearby locations are on the same cache line, serious performance degradation can result.

3.3.3.3 OpenMP MATVEC

This single-node, pure OpenMP implementation targets the Intel Knight's Landing architecture specifically, with the following aspects of the hardware in mind: (i) large number of threads and hyper-threading, (ii) very fast MCDRAM used in cache mode, and (iii) 512-bit SIMD vector units. In order to optimally use these features, an ordering of states was chosen in which the off-diagonal matrix elements form a series of diagonal bands, see Figure 3.1(a). The state ordering that produces these diagonal bands is simple: the states are simply sorted lexicographically, or equivalently, sorted by their values when the bitstrings are interpreted as integers. Iteration along these bands is very fast because the same operation is being applied repeatedly to neighboring data values. This makes data access patterns easily predictable for the prefetcher, and also allows easy vectorization with SIMD instructions.

Generally, when computing the off-diagonal elements, a given row index is converted to its corresponding state bitstring, that bitstring is manipulated to yield a new state bitstring, and that new state is converted back into a (column) index. That conversion from state back to index can cause performance issues. In general it can be performed in $\mathcal{O}(\log n)$ time using binary search, but that can become a large overhead since it needs to be performed for every matrix element that is computed. Instead, we compute the difference between the row and column indices directly, using the state. Recall that the only off-diagonal elements are flip-flop terms, which exchange two neighboring spins but do not affect any other part

of the state. Consider a row corresponding to a state

$$x_{L-1} \cdots x_{i+2}, 0, 1, x_{i-1} \cdots x_0,$$

where x_j represents the configuration of spin j. It will have a matrix element connecting it with the column corresponding to the state

$$x_{L-1} \cdots x_{i+2}, 1, 0, x_{i-1} \cdots x_0.$$

It can be shown that in a lexicographical ordering, the difference in indices between these two states is $\Delta = \binom{i}{\sum_{i=0}^{i-1} x_i}$, where $\binom{n}{k}$ is the binomial coefficient function.

In order to optimize for the prefetcher and for vectorization, we would like to operate on many sequential values in memory, which this reordering allows us to do. However, we would also like to compute all of the matrix elements for a single row of the matrix at the same time, in order to optimize for temporal locality of write operations to the same location in the output vector, such that the data can be stored in the cache in between writes. We can optimally balance these needs by iterating across small *blocks* of rows, of a size that corresponds to the L1 cache size. This allows us to iterate efficiently along each of the sequential elements in a particular block while not losing relevant data from the cache. Each OpenMP thread owns a unique set of rows of the matrix, corresponding to several of these blocks, and thus there is no concern about race conditions or need for atomic operations.

3.3.3.4 MPI–OpenMP MATVEC

For the hybrid MPI–OpenMP implementation, we use one MPI rank per node, and OpenMP for on-node parallelism. In this case, communication bandwidth is limiting for anything more than a few MPI ranks. Thus, we choose a state ordering that minimizes the amount of data that needs to be communicated. To do so, we employ a breadth-first-search based ordering strategy, that is reminiscent of Cuthill–McKee reordering [110]. We begin with some initial state, and perform a breadth-first search (BFS) through the graph corresponding to the Hamiltonian, recording basis states as we encounter them. Due to the structure of the Hamiltonian, this ordering greatly reduces the bandwidth of the matrix, as can be seen in Figure 3.1(b). This narrow bandwidth allows communication with only neighboring ranks in a linear topology, for up to $\sim$50 nodes.

This reordering does not permit the fast direct calculation of differences between indices that was used in the pure OpenMP version, though that same method could be used along with a lookup table. While this is ostensibly concerning, the index lookup time is ultimately irrelevant because it only affects the speed of the computational portion of the matrix-vector multiply. The communication is the dominant cost and the communication and computation are overlapped, so there is no overhead from a slightly slower computational portion.

3.4 Numerical Experiments

All numerical experiments were performed on the NERSC super computer called Cori which has 2 different types of compute nodes:

- Intel Xeon "Haswell" compute nodes @2.3 GHz, 2x16 cores and each 2 hyper-threads, 128 GB DDR4 RAM.
- Intel Xeon Phi "Knights Landing" (KNL) compute nodes @1.4 GHz, 68 cores and each with 4 hyper-threads, 96 GB DDR4 RAM, 16 GB MCDRAM.

3.4.1 OpenMP MATVEC

The full OpenMP, single node implementation was used to solve instances of the problem up to $L = 24$ spins. In this range of system sizes the required memory can easily fit on a single node, and even fully in the MCDRAM of a single KNL node, see Table 3.1.

Figure 3.2 shows the parallel speedup of the block MATVEC with block size 32 on both a Haswell and KNL node. The speedup is calculated as the ratio of running the OpenMP MATVEC code using the full number of available threads including hyper-threading, 64 threads and 272 threads, respectively, and using only a single thread. As can be seen in this figure, the implementation makes full use of the many-core architecture of the KNL nodes. At smaller system sizes, there is not quite enough work for each core to do to allow full utilization, but as the system size grows, all of the physical cores become well-utilized.

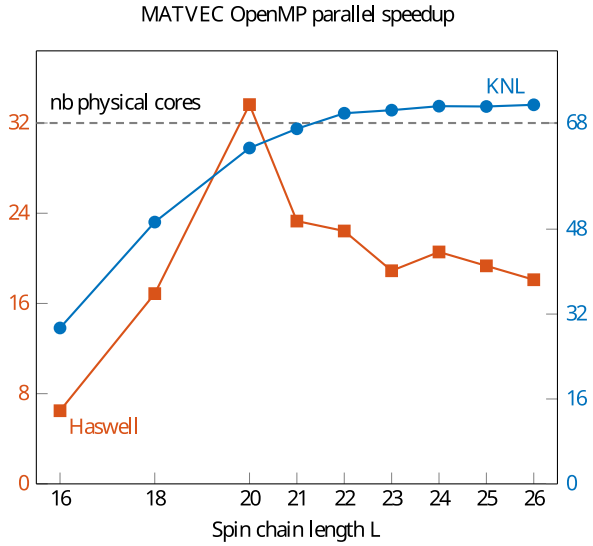


Figure 3.2: **OpenMP parallel speedup.** Here we present results for Haswell and KNL, for the block MATVEC with block size 32. The vertical axes correspond to the speedup obtained when running using the full number of available threads, when compared to running with only a single thread on the same hardware.

From a priori estimation and empirical evidence, it is clear that memory bandwidth is the limiting factor for the pure OpenMP matrix-vector multiplication. We hypothesize that competition for memory bandwidth is what prevents the Haswell nodes from efficiently using all the cores. On the Haswell nodes the 32 cores are split into two 16-core sockets, with each socket having four DIMMs. This means that four cores are using each DIMM concurrently, and it seems that the DRAM simply can't supply data fast enough to keep the CPUs saturated.

On KNL, however, the (more numerous) cores have both a lower individual clock rate and access to extremely fast 16 GB of MCDRAM with 460 GB/s total bandwidth. Our results show that this hardware, along with the matrix-vector multiplication designed for efficient vectorization, is able to keep all 68 cores supplied with data. In Figure 3.2 we also note that the speedup is actually slightly higher than the number of physical cores. We hypothesize that this is due to L1 and L2 cache effects, i.e., with less work per core in the parallel case, it is more likely that requested memory location will already be in the cache. Hence, for the remainder of the paper we will use the KNL compute nodes for all numerical experiments.

3.4.2 MPI–OpenMP MATVEC

For the hybrid MPI–OpenMP, we have implemented 3 different communication mechanisms: blocking using `MPI_Send/Recv`, non-blocking using `MPI_Isend/Irecv`, and one-sided remote memory access (rma) using `MPI_Put`. For the former ones we have also implemented 2 variants: one with and one without overlapping communication and local computation. The overlapping is achieved by explicitly allocating one OpenMP thread to the MPI calls, while the other threads perform the matrix-vector multiplication on the local matrix elements.

The results from a strong scaling experiment for all different variants of the MPI–OpenMP MATVEC with $L = 26$ and block size 64 are shown in Figure 3.3. The top figures clearly indicate that in this hybrid MPI-OpenMP case, the efficiency of the matrix-vector product is ultimately limited by communication bandwidth since overlapping communication and local computation yields a reduction of the wall time by 18% up to 36% in this case.

In Figure 3.3 we also note that the different communication mechanism result in very similar timing results. This is probably due to the predictable nature of the communication, i.e., neighboring nodes only and at predictable times. Overall the non-blocking `MPI_Isend/Irecv` implementation was found to be the most performant.

3.4.3 Eigenstate Entanglement Entropy

Using the pure OpenMP implementation up to system sizes of $L = 24$, we present preliminary data on the entanglement entropy for the localization transition. The performance data for these runs is given in Table 3.2.

For each disorder realization, 16 eigenpairs with eigenvalues nearest zero were computed, and the half-chain entanglement entropy was computed for each eigenvector. As is described in Section 3.2.2.2, the entropy in the thermal case in this regime is expected to scale linearly

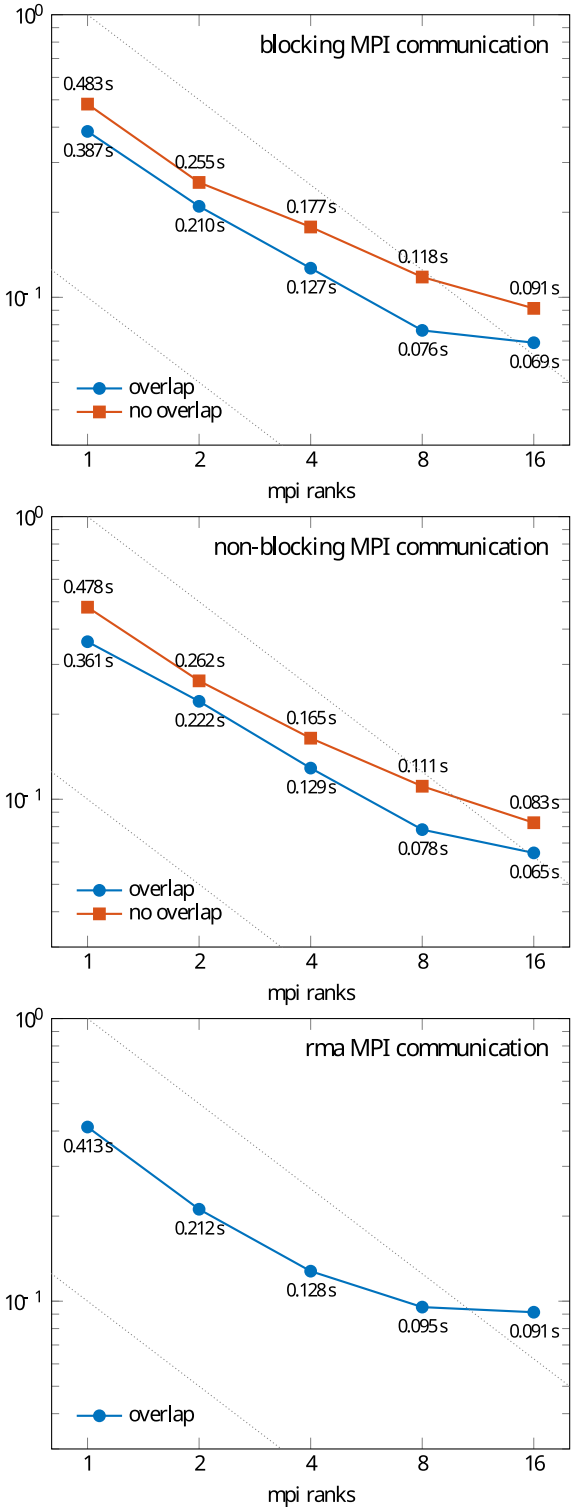


Figure 3.3: **Strong scaling.** Results are presented for the $L = 26$ MPI–OpenMP MATVEC and block size 64 with blocking, non-blocking, and rma MPI communication.

with L, with a constant correction of -0.5. To thus normalize the entropy across different system sizes we defined a *scaled entropy* simply as

$$\tilde{S} = \frac{S_{L/2} + 1/2}{L}.$$

In Figure 3.4, we plot the results of our computations of this scaled entropy. In general, we expect the scaled entropy to attain a value of $\ln 2$ in the thermal state for all system sizes, and then to decrease as the system becomes localized. This corresponds with what we see in our results, up to some small negative corrections at low system sizes, which were also seen in [62], though that paper used a slightly different metric.

When plotting the variance of the entanglement entropy in Figure 3.5, we see a clear description of the transition. At low disorder, every system size shows an exponential increase towards the transition, manifested as a line in the log-scaled variance plot. This exponential growth starts surprisingly early, and the slope of the growth in log scale, and thus the factor in the exponent, is consistent across system sizes. This can be interpreted as a consistent exponential approach to the transition, with the larger system sizes simply starting at smaller values, with the starting value changing by a constant factor with each increase in system size.

After this exponential growth, we see the curves peak, at a point which can be interpreted as the center of the transition. Our three largest system sizes qualitatively converge on a consistent point for this peak. With data for larger system sizes, we hope to be able to resolve this point precisely. A clear next step is to run full disorder averaging at a full set of disorder points for $L = 24$ and beyond. Timing results for $L = 26$ are given in Table 3.3. At these system sizes, the exponential growth region should converge with the observed peak, yielding insight into precisely where the transition occurs.

L	LOBPCG tolerance	Mean wall time
12	10^{-5}	0.6 s
14	10^{-5}	1.4 s
16	10^{-5}	4.5 s
18	10^{-5}	30.7 s
20	10^{-5}	4.7 min
22	10^{-5}	1.0 h
22	10^{-6}	1.2 h
24	10^{-6}	16.6 h

Table 3.2: Runtime to solve for the 16 eigenpairs with eigenvalues nearest to 0. Runs used a single KNL node with the full OpenMP solver implementation.

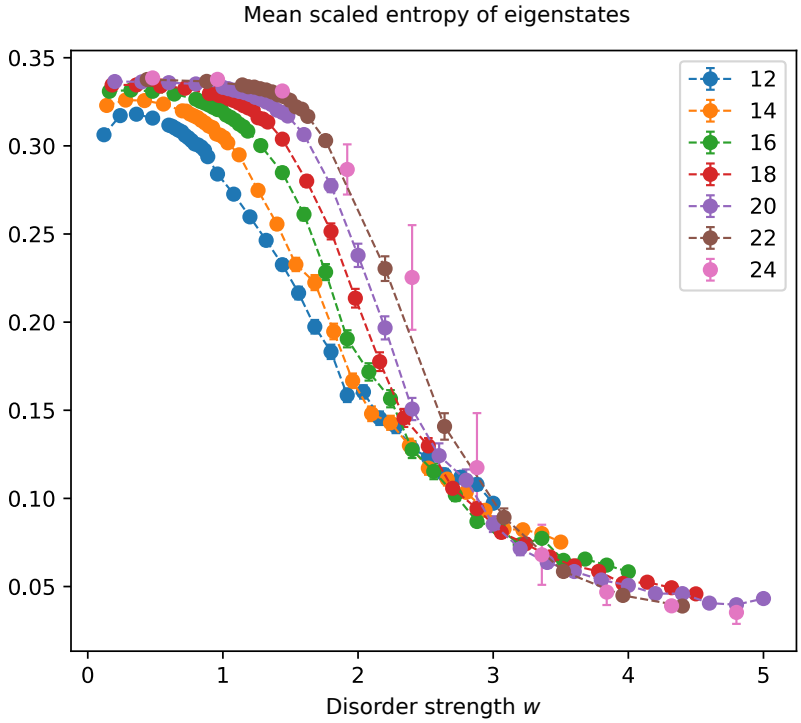


Figure 3.4: **Scaled half-chain entanglement entropy as a function of disorder strength.** We observe clear flow with increasing L.

Another next step is to do this same analysis but further away from the middle of the spectrum. Since our solver is faster in that region where the eigenvalues are less dense, as reported in Table 3.3, we should be able to observe the transition behavior at larger system sizes. Such a speedup is not possible with the shift-and-invert Lanczos algorithm, whose memory usage is ignorant to the value of the shift applied.

3.5 Conclusions

We have introduced a new approach to study many-body localization, which requires computing many eigenvalues and corresponding eigenvectors of large Hamiltonians in different regions of the spectrum. Our approach uses the LOBPCG algorithm, in combination with an efficient and matrix-free implementation of the block matrix-vector multiplication on many-core architectures to compute the desired eigenvalues and eigenvectors. Such an approach allows us to overcome the memory bottleneck in the previously used shift-and-invert Lanczos algorithm. As a result, the total memory footprint is reduced by several orders of magnitude, which allows us to compute eigenpairs of spin chains with up to $L = 24$ on a single compute node. We have also developed an hybrid MPI–OpenMP version of the solver that

can be run on several nodes. Because the MBL study requires solving eigenvalue problems for many instances of Hamiltonians with random disorder terms, and computing eigenvalues from different regions of the spectrum, the overall computation can scale to hundreds of thousands of computational cores.

ε	LOBPCG iter	PCG iter	time [h]
0.1	18	200	0.15
0.2	11	5,000	2.36
0.3	33	10,000	13.94
0.4	32	20,000	26.05
0.5	30	20,000	24.84
0.6	32	20,000	26.05
0.7	36	10,000	15.36
0.8	10	5,000	2.09
0.9	14	200	0.12

Table 3.3: **Timing results.** Here we present timings for computing 32 eigenpairs of $L = 26$ Hamiltonians as a function of the normalized shift ε. Runs used 32 KNL nodes and LOBPCG with block size 64, tolerance 10^{-6}, and preconditioned with PCG.

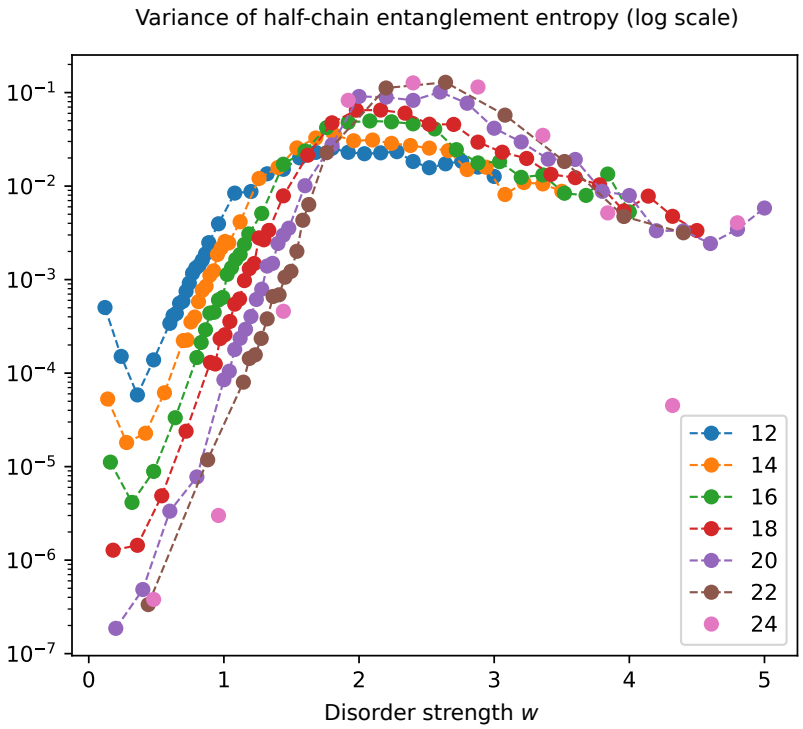


Figure 3.5: **Variance of the half-chain entanglement entropy as a function of disorder strength.** Again, we observe clear flow with the spin chain length L.

Part II

Cryptographic tests of quantum advantage

The primary goal of quantum computing is to perform certain computations faster (or "better" by some other metric) than classical computers, a notion which has been termed "quantum advantage." The pursuit of this goal has been complicated by two main challenges. The first is that individual, precise, and coherent control of individual atoms (or other quantum objects) is extremely difficult. The second is that, as we saw in the previous Part, modern classical computers are extremely powerful, and can simulate quantum systems of nontrivial size. This begs the question: what is the *simplest* computation one can do on a quantum computer, that is hard or impossible for today's best classical computers? While a seemingly simple question, it is steeped in subtlety. For example, if a noisy quantum computer produces some output that would be infeasible for a classical computer to reproduce, how do we reliably check that that output is actually *correct*? On the other hand, how do we ensure that the supposedly hard computational problem is *actually* hard for classical computers, rather than just sufficiently obscure that nobody had yet come up with a fast classical algorithm for it?

In this Part, we pursue these questions, with a focus on the *verifiability* of tests of quantum computational power, and how ideas from cryptography can be used to achieve these goals. In Chapter 4, we examine an efficiently classically verifiable test based on quantum sampling, which was first proposed in 2008, and break the underlying cryptography, showing that there is actually an algorithm by which a classical impostor can "forge" the results and impersonate a quantum device. This invalidates the test. In Chapter 5, we propose a new efficiently-verifiable test of quantumness whose classical hardness is *provably* as hard as factoring numbers. Factoring is arguably the computational problem which has received the most research effort of any to try to find an efficient classical algorithm, giving good confidence that the problem truly is classically hard. Furthermore, this new protocol can be implemented with fewer quantum resources than Shor's algorithm (the obvious way to use factoring to demonstrate quantum power), making it more amenable to near-term devices. In Chapter 6 we present a "proof-of-concept" experimental implementation of that new protocol, and another related one, in a trapped-ion quantum computer. From a technical perspective, the main innovation is the implementation of quantum measurements in the middle of a quantum circuit, followed by further quantum gates, while maintaining high fidelity. Such mid-circuit measurements have been a long-standing goal of experimental quantum computing, and are important not only for cryptographic protocols like the ones implemented here, but also a number of other applications such as feed-forward error correction. Finally, in Chapter 7, we present a novel method for performing coherent integer multiplication on quantum computers, which is a crucial operation for the both cryptographic protocols described earlier, and also the implementation of Shor's algorithm. Our method both improves on the asymptotic cost (in terms of gate count) of implementing multiplication, and simultaneously reduces practical overheads.

Chapter 4

Forging quantum data: classically defeating an IQP-based quantum test

4.1 Introduction

Recent experiments have demonstrated groundbreaking quantum computational power in the laboratory, showing *quantum computational advantage* [32]–[35]. In the past decade, much theoretical work has gone into designing experimental protocols expressly for this purpose, and providing evidence for the classical hardness of reproducing the experimental results [15], [17]–[21], [23], [39], [41], [111]–[115]. A difficulty with many of them, however, is that the quantum machine's output is hard to *verify*. In many cases, the best known algorithm for directly checking the solution is equivalent to classically performing the computational task itself. This presents challenges for validation of the test's results, because an ideal demonstration of quantum advantage occurs in the regime where a classical solution is not just difficult, but impossible with current technology. In that regime, experiments have had to resort to indirect methods to demonstrate that their devices are producing correct results [32]–[35].

In 2008, an efficiently-verifiable test of quantum computational advantage was proposed based on "instantaneous quantum polynomial-time" (IQP) circuits—quantum circuits in which all operations commute [116]. The protocol places only moderate requirements on the quantum device, making it potentially a good candidate for near-term hardware. Furthermore, later papers showed based on reasonable assumptions that classically sampling from the resulting distribution should be hard [16], [36]. This suggests that a "black-box" approach to cheating classically (by simply simulating the quantum device) is indeed computationally hard, and only a couple hundred qubits would be required to make a classical solution intractable.

Importantly, however, the classical verifier of the efficiently-verifiable protocol does not explicitly check whether the prover's samples come from the correct distribution (in fact, doing such a check efficiently is probably not possible [36]). Instead, the sampling task is

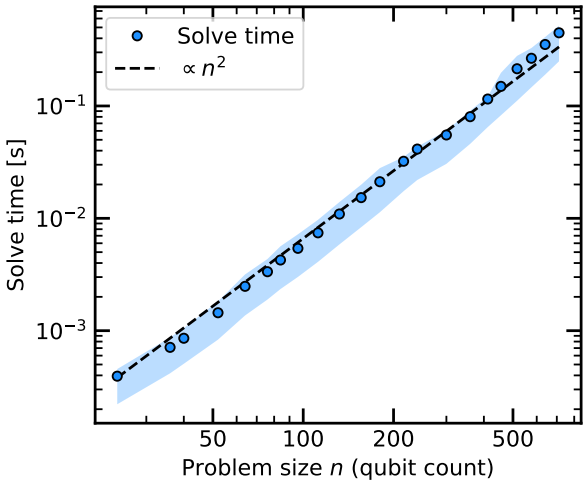


Figure 4.1: **Algorithm runtime.** Mean time to extract the secret vector $\boldsymbol{s}$ from X-programs constructed as described in [116]. Shaded region is the first to third quartile of the distribution of runtimes. We observe that the time is polynomial and fast in practice even up to problem sizes of hundreds of qubits. See Section 4.3.2 for a discussion of the $\mathcal{O}(n^2)$ scaling. The data points were computed by applying the algorithm to 1000 unique X-programs at each problem size. The secret vector was successfully extracted for every X-program tested. Experiments were completed using one thread on an Intel 8268 "Cascade Lake" processor.

designed such that bitstrings from its distribution will be orthogonal to some secret binary vector $\boldsymbol{s}$ with high probability, and it is this property that is checked by the verifier. A question that has remained open is whether a classical machine can efficiently generate samples satisfying the orthogonality check, without necessarily approximating the actual circuit's distribution. In this work we show that the answer to this question is yes. We give an explicit algorithm that can extract the secret bistring $\boldsymbol{s}$ underlying any instance of the protocol, thus making it trivial to generate orthogonal samples that pass the verifier's test. The main results described here are a statement of the algorithm, a proof that a single iteration of it will extract the secret vector $\boldsymbol{s}$ with probability $1/2$ (which can be made arbitrarily close to 1 by repetition), and empirical results demonstrating that the algorithm is efficient in practice (summarized in Figure 4.1).

The following is a summary of the paper's structure. In Section 4.2, we review the protocol's construction and some relevant analysis from the original paper. In Section 4.3 we describe the algorithm to extract the secret key, and therefore break the protocol's security against classical provers. There we also discuss briefly our implementation of the algorithm. In Section 4.4 we discuss the results, and provide the secret key underlying the "$25 challenge" that accompanied the publication of the protocol.

4.2 Background

Overview of protocol Here we summarize the IQP-based protocol for quantum advantage, in the standard cryptographic terms of an ostensibly quantum *prover* attempting to prove its quantum capability to a classical *verifier*. We refer the reader to the work that proposed the protocol for any details not covered here [116]. The core of the protocol is a sampling problem. The verifier generates a Hamiltonian H_P consisting of a sum of products of Pauli X operators, and asks the quantum prover to generate samples by measuring the state $e^{iH_P\theta}|0^{\otimes n}\rangle$ for some value of the "action" θ. The Hamiltonian H_P is designed such that the measured bitstrings $\{\boldsymbol{x}_i\}$ are biased with respect to a secret binary vector $\boldsymbol{s}$, so that $\boldsymbol{x}_i \cdot \boldsymbol{s} = 0$ with high probability (where $(\cdot)$ represents the binary inner product, modulo 2). The classical verifier, with knowledge of $\boldsymbol{s}$, can quickly check that the samples have such a bias. Since $\boldsymbol{s}$ should be only known to the verifier, it was conjectured that the only efficient way to generate such samples is by actually computing and measuring the quantum state [116]. However, in Section 4.3 we show that it is possible to extract $\boldsymbol{s}$ classically from just the description of the Hamiltonian.

X-programs A Hamiltonian of the type used in this protocol can be described by a rectangular matrix of binary values, for which each row corresponds to a term of the Hamiltonian. Given such a binary matrix P (called an "X-program"), the Hamiltonian is

$$H_P = \sum_i \prod_j X^{P_{ij}} \tag{4.1}$$

In words, a 1 in P at row i and column j corresponds to the inclusion of a Pauli X operator on the j^{th} site in the i^{th} term of the Hamiltonian. The X-program also has one additional parameter θ, which is the "action"—the integrated energy over time for which the Hamiltonian will be applied. For the protocol relevant to this work, the action is set to $\theta = \pi/8$ (see below).

Embedding a bias and verifying the output In order to bias the output distribution along $\boldsymbol{s}$, a submatrix with special properties is embedded within the matrix P. Notationally, for a vector $\boldsymbol{s}$ and matrix P, let the submatrix $P_{\boldsymbol{s}}$ be that which is generated by deleting all rows of P that are orthogonal to $\boldsymbol{s}$. Letting $\boldsymbol{X}$ represent the distribution of measurement results for a given X-program, it can be shown that the probability that a measurement outcome is orthogonal to the vector $\boldsymbol{s}$, $\Pr[\boldsymbol{X} \cdot \boldsymbol{s} = 0]$, depends only on the submatrix $P_{\boldsymbol{s}}$. The rows of P that are orthogonal to $\boldsymbol{s}$ are irrelevant. The protocol uses that fact to attempt to hide $P_{\boldsymbol{s}}$ (and thus $\boldsymbol{s}$): starting with a matrix $P_{\boldsymbol{s}}$ that produces a bias, we may attempt to hide it in a larger matrix P by appending rows that are random aside from having $\boldsymbol{p} \cdot \boldsymbol{s} = 0$, and then scrambling the new, larger matrix in a way that preserves the bias.

But what matrix $P_{\boldsymbol{s}}$ should one start with? In the protocol, the verifier sets $P_{\boldsymbol{s}}$ to the generator matrix for a binary code of block length $q \equiv 7 \pmod 8$ whose codewords $\boldsymbol{c}$ have

$\mathrm{wt}(\boldsymbol{c}) \in \{-1, 0\} \pmod 4$ (and both those weights are represented, that is, the codewords do not all have weight 0 (mod 4)). In [116], the authors suggest specifically using a binary quadratic residue (QR) code because it has the desired codeword weights. The action θ is set to $\pi/8$. As described in Facts 1 and 2 below, this configuration leads to a gap between the quantum and classical probabilities of generating samples orthogonal to $\boldsymbol{s}$ (for the best known classical strategy before this work). The verifier's check is then simply to request a large number of samples, and determine if the fraction orthogonal to $\boldsymbol{s}$ is too large to have likely been generated by any classical strategy.

In the two Facts below, we recall the probabilities corresponding to the quantum strategy and previously best-known classical strategy [116]. The reasoning behind the classical strategy (Fact 2) forms the setup for the new algorithm described in this paper; it is worth understanding its proof before moving on to the algorithm in Section 4.3.

Fact 1. ***Quantum strategy***

Let P be an X-program constructed by embedding a submatrix with the properties described above. Let $\boldsymbol{X}$ be a random variable representing the distribution of bitstrings from an n-qubit quantum state $e^{iH_P\pi/8}|0\rangle$ measured in the Z basis, where H_P is defined as in Equation 4.1. Then,

$$\Pr\left[\boldsymbol{X} \cdot \boldsymbol{s} = 0\right] = \cos^2\left(\frac{\pi}{8}\right) \approx 0.85\cdots \tag{4.2}$$

Proof. The entire proof is contained in [116]. To summarize, it is shown that for any string $\boldsymbol{z}$ and corresponding submatrix $P_{\boldsymbol{z}}$, the probability is

$$\Pr\left[\boldsymbol{X} \cdot \boldsymbol{z} = 0\right] = \mathop{\mathbb{E}}_{\boldsymbol{c}}\left[\cos^2\left(\theta \cdot (q - 2\mathrm{wt}(\boldsymbol{c})\right)\right] \tag{4.3}$$

where θ is the action, q is the number of rows in $P_{\boldsymbol{z}}$ and the expectation is taken over the codewords $\boldsymbol{c}$ of the code generated by the submatrix $P_{\boldsymbol{z}}$. When the values of $\theta = \pi/8$, $q \equiv 7 \pmod 8$ and $\mathrm{wt}(\boldsymbol{c}) \in \{-1, 0\} \pmod 4$ corresponding to the specific submatrix $P_{\boldsymbol{s}}$ are substituted into this expression, the result is Equation 4.2.

□

Fact 2. ***Classical strategy of [116]***

Again let P be an X-program constructed by embedding a submatrix with the properties described above. Let $\boldsymbol{d}, \boldsymbol{e}$ be two bitstrings of length n (the length of a row of P). Define $P_{\boldsymbol{d},\boldsymbol{e}}$ as the matrix generated by deleting the rows of P orthogonal to $\boldsymbol{d}$ or $\boldsymbol{e}$. [1] *Let $\boldsymbol{y} = \sum_{\boldsymbol{p}_i \in P_{\boldsymbol{d},\boldsymbol{e}}} \boldsymbol{p}_i$ be the vector sum of the rows of $P_{\boldsymbol{d},\boldsymbol{e}}$. Letting $\boldsymbol{Y}$ be the random variable representing the distribution of $\boldsymbol{y}$ when $\boldsymbol{d}$ and $\boldsymbol{e}$ are chosen uniformly at random, then*

$$\Pr\left[\boldsymbol{Y} \cdot \boldsymbol{s} = 0\right] = 3/4 \tag{4.4}$$

[1] In [116], $P_{\boldsymbol{d},\boldsymbol{e}}$ is written as $P_{\boldsymbol{d}} \cap P_{\boldsymbol{e}}$.

Proof. (From [116]) With $\boldsymbol{y}$ defined as above, we have

$$\boldsymbol{y} \cdot \boldsymbol{s} = \sum_{\boldsymbol{p}_i \in P_{\boldsymbol{d},\boldsymbol{e}}} \boldsymbol{p}_i \cdot \boldsymbol{s} \tag{4.5}$$

By defintion, $\boldsymbol{p}_i \cdot \boldsymbol{s} = 1$ if $\boldsymbol{p}_i \in P_{\boldsymbol{s}}$. Therefore $\boldsymbol{y} \cdot \boldsymbol{s}$ is equivalent to simply counting the number of rows common to $P_{\boldsymbol{s}}$ and $P_{\boldsymbol{d},\boldsymbol{e}}$, or equivalently, counting the rows in $P_{\boldsymbol{s}}$ for which $\boldsymbol{p} \cdot \boldsymbol{d}$ and $\boldsymbol{p} \cdot \boldsymbol{e}$ are both 1. We can express this using the matrix-vector products of $P_{\boldsymbol{s}}$ with $\boldsymbol{d}$ and $\boldsymbol{e}$:

$$\boldsymbol{y} \cdot \boldsymbol{s} = \sum_{\boldsymbol{p}_i \in P_{\boldsymbol{s}}} (\boldsymbol{p} \cdot \boldsymbol{d})\,(\boldsymbol{p} \cdot \boldsymbol{e}) \tag{4.6}$$

$$= (P_{\boldsymbol{s}}\ \boldsymbol{d}) \cdot (P_{\boldsymbol{s}}\ \boldsymbol{e}) \tag{4.7}$$

Considering that $P_{\boldsymbol{s}}$ is the generator matrix for an error correcting code, denote $\boldsymbol{c}_{\boldsymbol{d}} = P_{\boldsymbol{s}}\ \boldsymbol{d}$ as the encoding of $\boldsymbol{d}$ under $P_{\boldsymbol{s}}$. Then we have

$$\boldsymbol{y} \cdot \boldsymbol{s} = \boldsymbol{c}_{\boldsymbol{d}} \cdot \boldsymbol{c}_{\boldsymbol{e}} \tag{4.8}$$

Now, note that if a code has $\mathrm{wt}(\boldsymbol{c}) \in \{-1, 0\} \pmod 4$ for all codewords $\boldsymbol{c}$, the extended version of that code (created by adding a single parity bit) is doubly even, that is, has all codeword weights exactly 0 (mod 4). A doubly even binary code is necessarily self-dual, meaning all its codewords are orthogonal. This implies that any two codewords $\boldsymbol{c}_{\boldsymbol{d}}$ and $\boldsymbol{c}_{\boldsymbol{e}}$ of the original (non-extended) code have $\boldsymbol{c}_{\boldsymbol{d}} \cdot \boldsymbol{c}_{\boldsymbol{e}} = 0$ iff either $\boldsymbol{c}_{\boldsymbol{d}}$ or $\boldsymbol{c}_{\boldsymbol{e}}$ has even parity. Half of our code's words have even parity and $\boldsymbol{c}_{\boldsymbol{d}}$ and $\boldsymbol{c}_{\boldsymbol{e}}$ are random codewords, so the probability that either of them has even parity is $3/4$. Thus, the probability that $\boldsymbol{y} \cdot \boldsymbol{s} = 0$ is $3/4$, proving the fact. □

In the next section, we show that the classical strategy just described can be improved.

4.3 Algorithm

The classical strategy described in Fact 2 above generates vectors that are orthogonal to $\boldsymbol{s}$ with probability $3/4$. The key to classically defeating the protocol is that it is possible to *correlate* the vectors generated by that strategy, such that there is a non-negligible probability of generating a large set of vectors that *all* are orthogonal to $\boldsymbol{s}$. These vectors form a system of linear equations that can be solved to yield $\boldsymbol{s}$. Finally, with knowledge of $\boldsymbol{s}$ it is trivial to generate samples that pass the verifier's test.

We follow a modified version of the classical strategy of Fact 2 to generate each vector in the correlated set. Crucially, instead of choosing random bitstrings for both $\boldsymbol{d}$ and $\boldsymbol{e}$ each time, we generate a single random bitstring $\boldsymbol{d}$ and hold it constant, only choosing new random values for $\boldsymbol{e}$ with each iteration. If the encoding $\boldsymbol{c}_{\boldsymbol{d}}$ of $\boldsymbol{d}$ under $P_{\boldsymbol{s}}$ has even parity,

Algorithm 1 ExtractKey(P)
The algorithm to extract the secret vector $\boldsymbol{s}$ from an X-program P. n is the number of columns in the X-program, and $\xleftarrow{\$}$ means "select uniformly from the set."

1. Let $\boldsymbol{m}^* = \sum_{\boldsymbol{p} \in \text{rows}(P)} \boldsymbol{p}$.
2. Pick $\boldsymbol{d} \xleftarrow{\$} \{0,1\}^n$.
3. Generate a large number (say $2n$) of vectors $\boldsymbol{m}_i$ via the following steps, collecting the results into the rows of a matrix M.
   a) Pick $\boldsymbol{e} \xleftarrow{\$} \{0,1\}^n$
   b) Let $\boldsymbol{m}_i = \boldsymbol{m}^* + \sum_{\substack{\boldsymbol{p} \in \text{rows}(P) \\ \boldsymbol{p}\cdot\boldsymbol{d} = \boldsymbol{p}\cdot\boldsymbol{e} = 1}} \boldsymbol{p}$
4. Via linear solve, find the set of vectors $\{\boldsymbol{s}_i\}$ satisfying $M\boldsymbol{s}_i = \mathbf{1}$, where $\mathbf{1}$ is the vector of all ones.
5. For each candidate vector $\boldsymbol{s}_i$:
   a) Extract $P_{\boldsymbol{s}_i}$ from P by deleting the rows of P orthogonal to $\boldsymbol{s}_i$
   b) If adding a parity bit to each of the columns $\boldsymbol{c}$ of $P_{\boldsymbol{s}_i}$ yields the generator matrix for a code that is doubly even (all basis codewords are doubly even and mutually orthogonal), return $\boldsymbol{s}$ and exit.
6. No candidate vector $\boldsymbol{s}$ was found; return to step 2.

all of the generated vectors $\boldsymbol{m}_i$ will have $\boldsymbol{m}_i \cdot \boldsymbol{s} = 0$ (see Theorem 1 below). This occurs with probability $1/2$ over our choice of $\boldsymbol{d}$.

In practice, it is more convenient to do the linear solve if all $\boldsymbol{m}_i \cdot \boldsymbol{s} = 1$ instead of 0. This can be easily accomplished by adding to each $\boldsymbol{m}_i$ a vector $\boldsymbol{m}^*$ with $\boldsymbol{m}^* \cdot \boldsymbol{s} = 1$. It turns out that $\boldsymbol{m}^* = \sum_{\boldsymbol{p} \in \text{rows}(P)} \boldsymbol{p}$ has this property; see proof of Theorem 1.

The explicit algorithm for extracting the vector $\boldsymbol{s}$ is given in Algorithm 1.

4.3.1 Analysis

In this section we present a theorem and an empirical claim which demonstrate together that Algorithm 1 can be used to efficiently extract the key from any X-program constructed according to the protocol described in Section 4.2. The theorem shows that with probability $1/2$ a single iteration of the algorithm finds the vector $\boldsymbol{s}$. The empirical claim is that Algorithm 1 is efficient.

Theorem 1. *On input an X-program P containing a unique submatrix $P_{\boldsymbol{s}}$ with the properties described in Section 4.2, a single iteration of Algorithm 1 will output the vector $\boldsymbol{s}$ corresponding to $P_{\boldsymbol{s}}$ with probability $\frac{1}{2}$.*

Proof. If $\boldsymbol{s}$ is contained in the set $\{\boldsymbol{s}_i\}$ generated in step 4 of the algorithm, the correct vector $\boldsymbol{s}$ will be output via the check in step 5 because there is a unique submatrix $P_{\boldsymbol{s}}$ with codewords having $\mathrm{wt}(\boldsymbol{c}) \in \{-1, 0\} \pmod 4$. $\boldsymbol{s}$ will be contained in $\{\boldsymbol{s}_i\}$ as long as M satisfies the equation $M\boldsymbol{s} = \mathbf{1}$. Thus the proof reduces to showing that $M\boldsymbol{s} = \mathbf{1}$ with probability $1/2$.

Each row of M is

$$\boldsymbol{m}_i = \boldsymbol{m}^* + \bar{\boldsymbol{m}}_i \tag{4.9}$$

for a vector $\bar{\boldsymbol{m}}_i$ defined as

$$\bar{\boldsymbol{m}}_i = \sum_{\substack{\boldsymbol{p}\in\mathrm{rows}(P) \\ \boldsymbol{p}\cdot\boldsymbol{d}=\boldsymbol{p}\cdot\boldsymbol{e}=1}} \boldsymbol{p} \tag{4.10}$$

Here we will show that $\boldsymbol{m}^* \cdot \boldsymbol{s} = 1$ always and $\bar{\boldsymbol{m}}_i \cdot \boldsymbol{s} = 0$ for all i with probability $1/2$, implying that $M\boldsymbol{s} = \mathbf{1}$ with probability $1/2$.

First we show that $\boldsymbol{m}^* \cdot \boldsymbol{s} = 1$. $\boldsymbol{m}^*$ is the sum of all rows of P, so we have

$$\boldsymbol{m}^* \cdot \boldsymbol{s} = \sum_{\boldsymbol{p}\in\mathrm{rows}(P)} \boldsymbol{p}\cdot\boldsymbol{s} = \sum_{\boldsymbol{p}\in\mathrm{rows}(P_{\boldsymbol{s}})} 1 \tag{4.11}$$

We see that the inner product is equal to the number of rows in the submatrix $P_{\boldsymbol{s}} \pmod 2$. This submatrix is a generator matrix for a code of block size $7 \pmod 8$; thus the number of rows is odd and

$$\boldsymbol{m}^* \cdot \boldsymbol{s} = 1 \tag{4.12}$$

Now we turn to showing that $\bar{\boldsymbol{m}}_i \cdot \boldsymbol{s} = 0$ for all i with probability $1/2$. In the proof of Fact 2, it was shown that for any two vectors $\boldsymbol{d}$ and $\boldsymbol{e}$, vectors $\bar{\boldsymbol{m}}_i$ generated by summing rows $\boldsymbol{p}_i$ of P for which $\boldsymbol{d}\cdot\boldsymbol{p}_i = \boldsymbol{e}\cdot\boldsymbol{p}_i = 1$ have

$$\bar{\boldsymbol{m}}_i \cdot \boldsymbol{s} = 0 \text{ iff } \boldsymbol{c}_{\boldsymbol{d}} \text{ or } \boldsymbol{c}_{\boldsymbol{e}} \text{ has even parity} \tag{4.13}$$

where $\boldsymbol{c}_{\boldsymbol{d}}$ and $\boldsymbol{c}_{\boldsymbol{e}}$ are the encodings under $P_{\boldsymbol{s}}$ of $\boldsymbol{d}$ and $\boldsymbol{e}$ respectively. If $\boldsymbol{d}$ is held constant for all i, and $\boldsymbol{d}$ happened to be chosen such that $\boldsymbol{c}_{\boldsymbol{d}} = P_{\boldsymbol{s}}\,\boldsymbol{d}$ has even parity, then $\bar{\boldsymbol{m}}_i \cdot \boldsymbol{s} = 0$ for all i by Equation 4.13. Because half of the codewords have even parity, for $\boldsymbol{d}$ selected uniformly at random we have $\bar{\boldsymbol{m}}_i \cdot \boldsymbol{s} = 0$ for all i with probability $1/2$.

We have shown that $\boldsymbol{m}^* \cdot \boldsymbol{s} = 1$ always and $\bar{\boldsymbol{m}}_i \cdot \boldsymbol{s} = 0$ for all i with probability $1/2$. Therefore we have

$$\Pr_{\boldsymbol{d}}\left[\boldsymbol{m}_i \cdot \boldsymbol{s} = 1 \;\forall\; i\right] = 1/2$$

Thus $M\boldsymbol{s} = \mathbf{1}$ with probability $1/2$. The algorithm will output $\boldsymbol{s}$ whenever $M\boldsymbol{s} = \mathbf{1}$, proving the theorem. □

Before we move on, we remark that while Theorem 1 treats X-programs containing a single unique submatrix with the relevant properties, the algorithm can easily be modified to return the vectors $\boldsymbol{s}$ corresponding to *all* such submatrices, if more exist, by simply accumulating all vectors $\boldsymbol{s}$ for which the check in Step 5(b) succeeds. We do note, however, that for the protocol described in Section 4.2, the probability of "extraneous" submatrices other than the one intentionally built into the matrix arising by chance is vanishingly small—corresponding to the probability that a random binary linear code happens to be doubly even and self-dual, which is bounded from above by $1/4^n$.

Now, having established that each iteration of the algorithm outputs $\boldsymbol{s}$ with probability $1/2$, we now turn to analyzing its runtime.

Claim 1. *(empirical) Algorithm 1 halts in $\mathcal{O}(n^3)$ time on average.*

All steps of the algorithm except for step 5 have $\mathcal{O}(n^3)$ scaling by inspection. The obstacle preventing Claim 1 from trivially holding is that it is hard to make a rigorous statement about how large the set of candidate vectors $\{\boldsymbol{s}_i\}$ is. Because $|\{\boldsymbol{s}_i\}| = 2^{n-\mathrm{rank}(M)}$, we'd like to show that on average, the rank of M is close to or equal to n. It seems reasonable that this would be the case: we are generating the rows of M by summing rows from P, and P must have full rank because it contains a rank-n error correcting code. But the rows of P summed into each $\boldsymbol{m}_i$ are not selected independently—they are always related via their connection to the vectors $\boldsymbol{d}$ and $\boldsymbol{e}$, and it's not clear how these correlations affect the linear independence of the resulting $\boldsymbol{m}_i$.

Despite the lack of a proof, empirical evidence supports Claim 1 when the algorithm is applied to X-programs generated in the manner described in Section 4.2. Figure 4.2(a) shows the average number of candidate keys checked by the algorithm before $\boldsymbol{s}$ is found, as a function of problem size. The value is constant, demonstrating that the average size of the set $\{\boldsymbol{s}_i\}$ does not scale with n. Furthermore, the value is small—only about 4. This implies that M usually has high rank. In Figure 4.2(b) we plot explicitly the distribution of the rank of the matrix M over 1000 runs of the algorithm on unique X-programs of size $n = 245$. The blue bars (on the left of each pair) show the distribution over all X-programs tested, and the sharply decaying tail supports the claim that low-rank M almost never occur.

A natural next question is whether there is some feature of the X-programs in that tail that causes M to be low rank. To investigate that question, the algorithm was re-run 100 times on each of the X-programs that had $n - \mathrm{rank}(M) > 4$ in the blue distribution. The orange bars of Figure 4.2(b) (on the right of each pair) plot the distribution of $n - \mathrm{rank}(M)$ for that second run. The similarity of the blue and orange distributions suggests that the rank of M is not correlated between runs; that is, the low rank of M in the first run was not due to any feature of the input X-programs. From a practical perspective, this data suggests that if the rank of M is found to be unacceptably low, the algorithm can simply be re-run with new randomness and the rank of M is likely to be higher the second time.

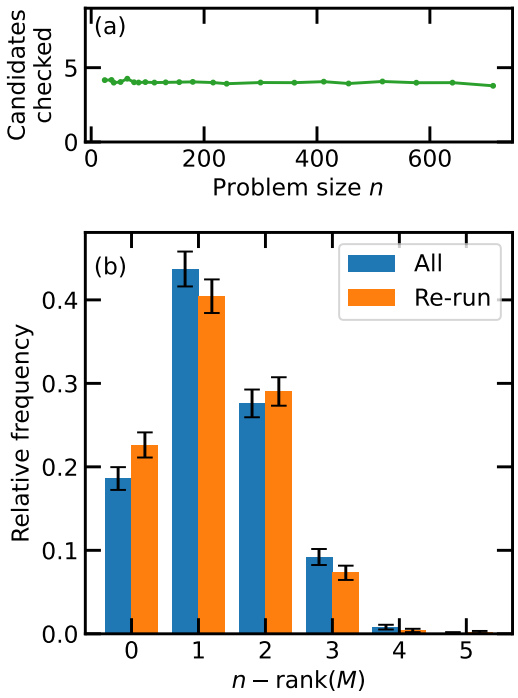


Figure 4.2: **Analysis of number of candidate vectors to be checked. (a)** The average number of candidate vectors checked before the secret vector $\boldsymbol{s}$ was found, when the algorithm was applied to 1000 unique X-programs at each problem size tested. We observe that the number of vectors to check is qualitatively constant in n. **(b)** The number of unconstrained degrees of freedom $n - \text{rank}\,(M)$ for matrices M generated in step 3 of Algorithm 1, for "good" choices of $\boldsymbol{d}$ such that $M\boldsymbol{s} = \boldsymbol{1}$. The rapidly decaying tail qualitatively implies that it is rare for any more than a few degrees of freedom to remain unconstrained. The blue bars represent the distribution over 1000 unique X-programs of size $n = 245$. The algorithm was then re-run on the X-programs that had $n - \text{rank}\,(M) > 4$ to generate the orange bars.

4.3.2 Implementation

An implementation of Algorithm 1 in the programming language Julia (along with the code to generate the figures in this manuscript) is available online [117]. Figure 4.1 shows the runtime of this implementation for various problem sizes. Experiments were completed using one thread on an Intel 8268 "Cascade Lake" processor.

Note that Figure 4.1 shows $\mathcal{O}\,(n^2)$ scaling, rather than $\mathcal{O}\,(n^3)$ from Claim 1. This is due to data-level parallelism in the implementation. $\mathbb{Z}_2^n$ vectors are stored as the bits of 64-bit integers, so operations like vector addition can be performed on 64 elements at once via bitwise operations. Furthermore, with AVX SIMD CPU instructions, those operations can be applied to multiple 64-bit integers in one CPU cycle. Thus, for n of order 100, the ostensibly $\mathcal{O}\,(n)$ vector inner products and vector sums are performed in constant time,

removing one factor of n from the runtime. The tests in Figure 4.1 were performed on a CPU with 512 bit vector units.

4.4 Discussion

Modifications to the protocol A natural question is whether it is possible to modify the original protocol such that this attack is not successful. Perhaps P can be engineered such that either 1) it is not possible to generate a large number of vectors that all have a known inner product with $\boldsymbol{s}$, or 2) the rank of the matrix M formed by these generated vectors will never be sufficiently high to allow solution of the linear system.

For 1), our ability to generate many vectors orthogonal to $\boldsymbol{s}$ relies on the fact that the code generated by the hidden submatrix $P_{\boldsymbol{s}}$ has codewords $\boldsymbol{c}$ with $\mathrm{wt}(\boldsymbol{c}) \in \{-1, 0\} \pmod 4$, as shown in the proof of Theorem 1. Unfortunately, this property regarding the weights of the codewords is precisely what gives the quantum sampling algorithm its bias toward generating vectors with $\boldsymbol{x} \cdot \boldsymbol{s} = 0$ (see Fact 1). This fact seems to preclude the possibility of simply removing the special property of the submatrix $P_{\boldsymbol{s}}$ to prevent the attack.

For 2), the main obstacle is that the matrix P *must* have rank n because embedded in it is a code of rank n. The only hope is to somehow engineer the matrix such that linear combinations generated in the specific way described above will not themselves be linearly independent. It is not at all clear how one would do that, and furthermore, adding structure to the previously-random extra rows of P runs the risk of providing even more information about the secret vector $\boldsymbol{s}$. Perhaps one could *prove* that the rank of M will be large even for worst-case inputs P—this could be an interesting future direction.

Protocols with provable hardness The attack described in this paper reiterates the value of building protocols for which passing the test itself, rather than just simulating the quantum device, can be shown to be hard under well-established cryptographic assumptions. In the past few years, a number of new trapdoor claw-free function based constructions have been proposed for demonstrating quantum computational advantage [39], [41], [115], [118], as well as some based on other types of cryptography [44], [45]. Unfortunately, such rigorous results come with a downside, which is an increase in the size and complexity of circuits that must be run on the quantum device. Exploring simplified protocols that are provably secure is an exciting area for further research.

The $25 challenge When the protocol was first proposed in [116], it was accompanied by an internet challenge. The authors posted a specific instance of the matrix P, and offered $25 to anyone who could send them samples passing the verifier's check. The secret vector $\boldsymbol{s}$ corresponding to their challenge matrix P is (encoded as a base-64 string):

```
BilbHzjYxrOHYH4OlEJFBoXZbps4a54kH8flrRgo/g==
```

The key was extracted using the implementation of Algorithm 1 described in Section 4.3.2.

Shepherd and Bremner, the authors of the challenge, have graciously confirmed that this indeed is the correct key.

Summary and outlook Here we have described a classical algorithm that passes the interactive quantum test described in [116]. We have proven that a single iteration of the algorithm will return the underlying secret vector with probability $1/2$, and empirically shown that it is efficient. The immediate implication of this result is that the protocol in its original form is no longer effective as a test of quantum computational power. While it may be possible to reengineer that protocol to thwart this attack, this paper reiterates the value of proving the security of the verification step. Furthermore, while protocols for quantum advantage with provable classical hardness are valuable in their own right, they can also be used as building blocks for achieving new, more complex cryptographic tasks, like certifiable random number generation, secure remote state preparation, and even the verification of arbitrary quantum computations [39], [47], [48]. As quantum hardware continues to improve and to surpass the abilities of classical machines, quantum cryptographic tools will play an important role in making quantum computation available as a service. Establishing the security of these protocols is an important first step.

Chapter 5

Classically-verifiable quantum advantage from a computational Bell test

5.1 Introduction

The development of large-scale programmable quantum hardware has opened the door to testing a fundamental question in the theory of computation: can quantum computers outperform classical ones for certain tasks? This idea, termed quantum computational advantage, has motivated the design of novel algorithms and protocols to demonstrate advantage with minimal quantum resources such as qubit number and gate depth [15]–[23], [111]. Such protocols are naturally characterized along two axes: the computational speedup and the ease of verification. The former distinguishes whether a quantum algorithm exhibits a polynomial or super-polynomial speedup over the best known classical one. The latter classifies whether the correctness of the quantum computation is *efficiently verifiable* by a classical computer. Along these axes lie three broad paths to demonstrating advantage: 1) sampling from entangled quantum many-body wavefunctions, 2) solving a deterministic problem, e.g. prime factorization, via a quantum algorithm, and 3) proving quantumness through interactive protocols.

Sampling-based protocols directly rely on the classical hardness of simulating quantum mechanics [15], [16], [20]–[23]. The "computational task" is to prepare and measure a generic complex many-body wavefunction with little structure. As such, these protocols typically require minimal resources and can be implemented on near-term quantum devices [32], [33]. The correctness of the sampling results, however, is exponentially difficult to verify. This has an important consequence: in the regime beyond the capability of classical computers, the sampling results cannot be explicitly checked, and quantum computational advantage can only be inferred (e.g. extrapolated from simpler circuits).

Algorithms in the second class of protocols are naturally broken down by whether they exhibit polynomial or super-polynomial speed-ups. In the case of polynomial speed-ups, there exist notable examples that are *provably* faster than any possible classical algorithm

[112], [113]. However, polynomial speed-ups are tremendously challenging to demonstrate in practice, due to the slow growth of the separation between classical and quantum runtimes [1]. Accordingly, the most attractive algorithms for demonstrating advantage tend to be those with a super-polynomial speed-up, including Abelian hidden subgroup problems such as factoring and discrete logarithms [119]. The challenge is that for all known protocols of this type, the quantum circuits required to demonstrate advantage are well beyond the capabilities of near-term experiments.

The final class of protocols demonstrates quantum advantage through an *interactive proof* [39], [41], [120]–[125]. At a high level, this type of protocol involves multiple rounds of communication between the classical verifier and the quantum prover; the prover must give self-consistent responses despite not knowing what the verifier will ask next. This requirement of self-consistency rules out a broad range of classical cheating strategies and can imbue "hardness" into questions that would otherwise be easy to answer. To this end, interactive protocols expand the space of computational problems that can be used to demonstrate quantum advantage; from a more pragmatic perspective, this can enable the realization of efficiently verifiable quantum advantage on near-term quantum hardware.

Recently, a beautiful interactive protocol was introduced that can operate both as a test for quantum advantage and as a generator of certifiable quantum randomness [39]. The core of the protocol is a two-to-one function, f, built on the computational problem known as learning with errors (LWE) [126]. The demonstration of advantage leverages two important properties of the function: first, it is *claw-free*, meaning that it is computationally hard to find a pair of inputs (x_0, x_1) such that $f(x_0) = f(x_1)$. [2]. Second, there exists a *trapdoor*: given some secret data t, it becomes possible to efficiently invert f and reveal the pair of inputs mapping to any output. (See Section 5.7.5 for an overview of trapdoor claw-free functions). However, to fully protect against cheating provers, the protocol requires a stronger version of the claw-free property called the *adaptive hardcore bit*, namely, that for any input x_0 (which may be chosen by the prover), it is computationally hard to find even a single bit of information about x_1 [3]. The need for an adaptive hardcore bit within this protocol severely restricts the class of functions that can operate as verifiable tests of quantum advantage.

Here, we propose and analyze a novel interactive quantum advantage protocol that removes the need for an adaptive hardcore bit, with essentially zero overhead in the quantum circuit and no extra cryptographic assumptions. We present four main results. First, we demonstrate how an idea from tests of Bell's inequality can serve the same cryptographic

[1] They also have some other caveats: a provable speedup of $\mathcal{O}(1)$ quantum complexity over $\mathcal{O}(n)$ classical complexity is promising, but just reading the input may require $\mathcal{O}(n)$ time, hiding the computational speedup in practice.

[2] "Claw-free" is often used to refer to a *pair* of functions f_0, f_1 such that for appropriate x_0, x_1 we have $f_0(x_0) = f_1(x_1)$. Here, we use the slightly more general idea of a single 2-to-1 function f for which it is hard to find x_0, x_1 such that $f(x_0) = f(x_1)$. This is a special case of a "collision-resistant function," which could potentially be many-to-one. We also note that a claw-free pair of functions can be converted into a single claw-free function by defining $f(b||x) = f_b(x)$, where $||$ denotes concatenation.

[3] To be precise, it is hard to find both x_0 and the parity of any subset of the bits of x_1.

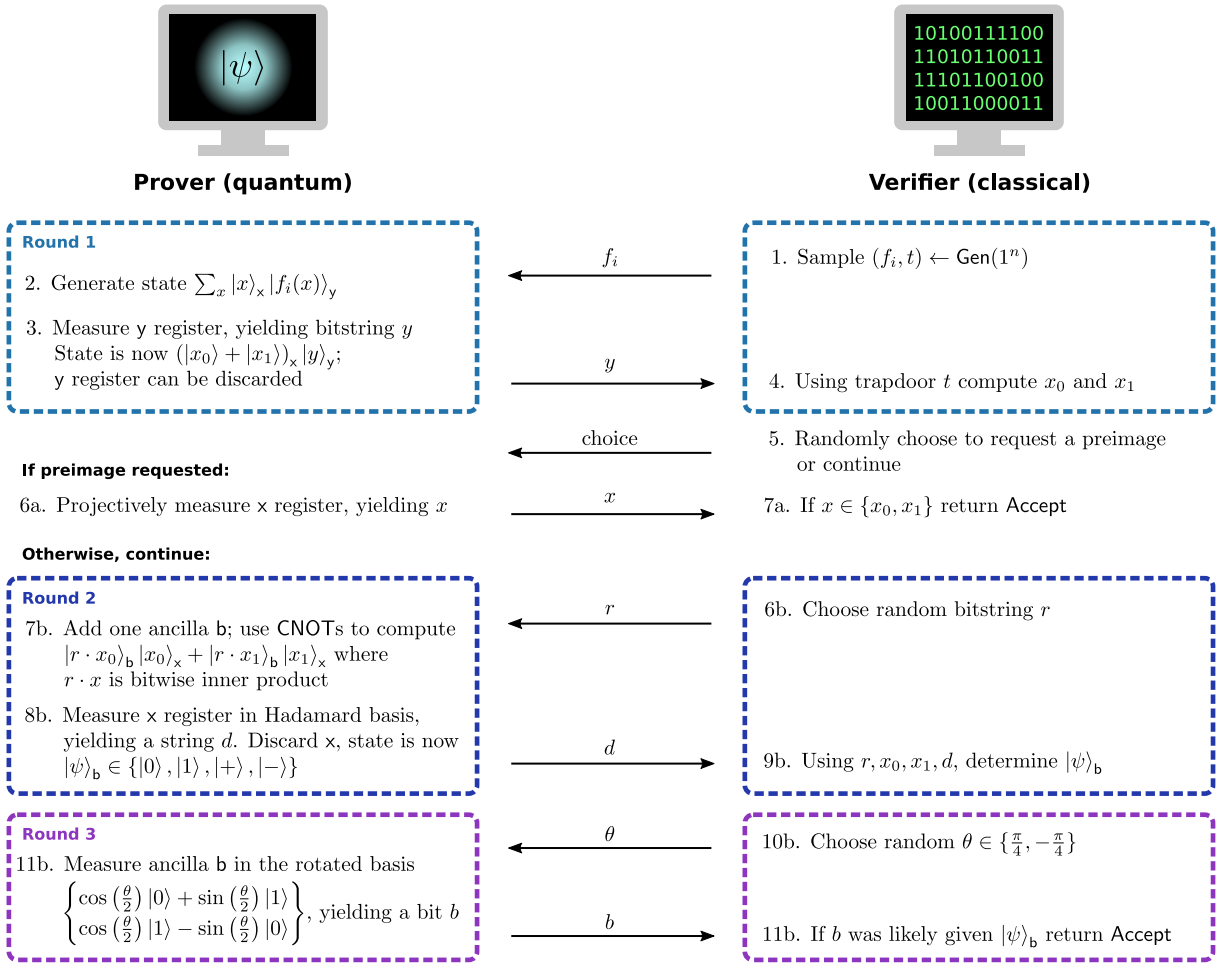


Figure 5.1: **Schematic representation of the interactive quantum advantage protocol.** In the first round of interaction, the classical verifier (right) selects a specific function from a trapdoor claw-free family and the quantum prover (left) evaluates it over a superposition of inputs. The goal of the second round is to condense the information contained in the prover's superposition state onto a single ancilla qubit. The final round of interaction effectively performs a Bell inequality measurement, whose outcome is cryptographically protected.

purpose as the adaptive hardcore bit [127]. In essence, our interactive protocol is a variant of the CHSH (Clauser, Horne, Shimony, Holt) game [128] in which one player is replaced by a cryptographic construction. Normally, in CHSH, two quantum parties are asked to produce correlations that would be impossible for classical devices to produce. If space-like separation is enforced to rule out communication between the two parties, then the correlations constitute a proof of quantumness. In our case, the space-like separation is replaced by the computational hardness of a cryptographic problem. In particular, the quantum prover holds a qubit whose state depends on the cryptographic secret in the same way that the state of one CHSH player's qubit depends on the secret measurement basis of the other player. An alternative interpretation, from the perspective of Bell's theorem, is that the protocol can be thought of as a "single-detector Bell test"—the cryptographic task generates the same single-qubit state as would be produced by entangling a second qubit and measuring it with another detector. As in the CHSH game, a quantum device can pass the verifier's test with probability $\sim 85\%$, but a classical device can only succeed with probability at most 75%. This finite gap in success probabilities is precisely what enables a verifiable test of quantum advantage.

Second, by removing the need for an adaptive hardcore bit, our protocol accepts a broader landscape of functions for interactive tests of quantum advantage (see Table 5.1 and Methods). We populate this list with two new constructions. The first is based on the decisional Diffie-Hellman problem (DDH) [129]–[131]; the second utilizes the function $f_N(x) = x^2 \bmod N$ with N the product of two primes, which forms the backbone of the Rabin cryptosystem [132], [133]. On the one hand, DDH is appealing because the elliptic-curve version of the problem is particularly hard for classical computers [134]–[136]. On the other hand, $x^2 \bmod N$ can be implemented significantly more efficiently, while its hardness is equivalent to factoring. We hope that these two constructions will provide a foundation for the search for more TCFs with desirable properties (small key size and efficient quantum circuits).

Third, we describe two innovations that facilitate our protocol's use in practice: a way to significantly reduce overhead arising from the reversibility requirement of quantum circuits, and a scheme for increasing noisy devices' probability of passing the test. Normally, quantum implementations of classical functions like the TCFs used in this protocol have some overhead, due to the need to make the circuit reversible in order to be consistent with unitarity [137]–[141]. Our protocol exhibits the surprising property that it permits a measurement scheme to discard so-called "garbage bits" that arise during the computation, allowing classical circuits to be converted into quantum ones with essentially zero overhead. In the case of a noisy quantum device, the protocol also enables an inherent post-selection scheme for detecting and removing certain types of quantum errors. With this scheme it is possible for quantum devices to trade off low quantum fidelities for an increase in the overall runtime, while still passing the cryptographic test. We note that these constructions are likely applicable to other TCF-based quantum cryptography protocols as well, and thus may be of independent interest for tasks such as certifiable quantum random number generation.

Finally, focusing on the TCF $x^2 \bmod N$, we provide explicit quantum circuits—both asymptotically optimal (requiring only $\mathcal{O}(n \log n)$ gates and $\mathcal{O}(n)$ qubits), as well as those

Problem	Trap door	Claw-free	Adaptive hard-core bit	Asymptotic complexity (gate count)
LWE [39]	✓	✓	✓	$n^2 \log^2 n$
$x^2 \bmod N$	✓	✓	✗	$n \log n$
Ring-LWE [41]	✓	✓	✗	$n \log^2 n$
Diffie-Hellman	✓	✓	✗	$n^3 \log^2 n$
Shor's alg.	—	—	—	$n^2 \log n$

Table 5.1: **Cryptographic constructions for interactive quantum advantage protocols.** n represents the number of bits in the function's input string. Big-$\mathcal{O}$ notation is implied an d factors of $\log \log n$ and smaller are dropped. For references and derivations of the circuit complexities, see Section 5.7.6.

aimed for near-term quantum devices. We show that a verifiable test of quantum advantage can be achieved with $\sim 10^3$ qubits and a gate depth $\sim 10^5$ (see Methods). We also co-design a specific implementation of $x^2 \bmod N$ optimized for a programmable Rydberg-based quantum computing platform. The native physical interaction corresponding to the Rydberg blockade mechanism enables the direct implementation of multi-qubit-controlled arbitrary phase rotations without the need to decompose such gates into universal two-qubit operations [142]–[146]. Access to such a native gate immediately reduces the gate depth for achieving quantum advantage by an order of magnitude.

5.2 Background and Related Work

The use of trapdoor claw-free functions for quantum cryptographic tasks was pioneered in two recent breakthrough protocols: (i) giving classical homomorphic encryption for quantum circuits [46] and (ii) for generating cryptographically certifiable quantum randomness from an untrusted black-box device [39]; this latter work also introduced the notion of an adaptive hardcore bit and serves as an efficiently verifiable test of quantum advantage. Remarkably, the scheme was further extended to allow a classical server to cryptographically verify the correctness of arbitrary quantum computations [47]; it has also been applied to remote state preparation with implications for secure delegated computation [48].

Recently, an improvement to the practicality of TCF-based proofs of quantumness was provided in the *random oracle model* (ROM)—a model of computation in which both the quantum prover and classical verifier can query a third-party "oracle," which returns a random (but consistent) output for each input. In that work, the authors provide a protocol that both removes the need for the adaptive hardcore bit, and also reduces the interaction to a single round [41]. Because the security of the protocol is proven in the ROM, imple-

menting this protocol in practice requires applying the *random oracle heuristic*, in which the random oracle is replaced by a cryptographic hash function, but the hardness of classically defeating the protocol is taken to still hold [4]. Only contrived cryptographic schemes have ever been broken by attacking the random oracle heuristic [42], [43], so it seems to be effective in practice and the ROM protocol has significant potential for use as a practical tool for benchmarking untrusted quantum servers. On the other hand, for a robust experimental test of the foundational complexity theoretic claims of quantum computing—that quantum mechanics allows for algorithms that are superpolynomially faster than classical Turing machines—we desire the complexity-theoretic backing of the speedup to be as strong as possible (i.e. provable in the "standard model" of computation [22]), which is the goal pursued in the present work. With that said, we emphasize that the various optimizations described below—including the TCF families based on DDH and x^2 mod N, as well as the schemes for postselection and discarding garbage bits—can be applied to the ROM protocol as well.

Lastly, we also note two recent works which demonstrate that any TCF-based proof of quantumness, including the present work, can be implemented in constant quantum circuit depth (at the cost of more qubits) [147], [148].

5.3 Interactive Protocol for Quantum Advantage

Our full protocol is shown diagrammatically in Figure 5.1. It consists of three rounds of interaction between the prover and verifier (with a "round" being a challenge from the verifier, followed by a response from the prover). The first round generates a multi-qubit superposition over two bit strings that would be cryptographically hard to compute classically. The second round maps this superposition onto the state of one ancilla qubit, retaining enough information to ensure that the resulting single-qubit state is also hard to compute classically. The third round takes this single qubit as input to a CHSH-type measurement, allowing the prover to generate a bit of data that is correlated with the cryptographic secret in a way that would not be possible classically. Having described the intuition behind the protocol, we now lay out each round in detail.

5.3.1 Description of the protocol

The goal of the first round is to generate a superposition over two colliding inputs to the trapdoor claw-free function (TCF). It begins with the verifier choosing an instance f_i of the TCF along with the associated trapdoor data t; f_i is sent to the prover. As an example, in the case of $x^2 \mod N$, the "index" i is the modulus N, and the trapdoor data is its

[4] Replacing the random oracle with a hash function is termed a *heuristic* rather than an *assumption* because the security of this procedure generally holds in practice but is not provable—in fact, there exist constructions that are provably secure in the random oracle model but trivially insecure when instantiated with a hash function [42].

factorization, p, q. The prover now initializes two registers of qubits, which we denote as the x and y registers. On these registers, they compute the entangled superposition $|\psi\rangle = \sum_x |x\rangle_{\mathsf{x}} |f_i(x)\rangle_{\mathsf{y}}$, over all x in the domain of f_i. The prover then measures the y register in the standard basis, collapsing the state to $(|x_0\rangle + |x_1\rangle)_{\mathsf{x}} |y\rangle_{\mathsf{y}}$, with $y = f(x_0) = f(x_1)$. The measured bitstring y is then sent to the verifier, who uses the secret trapdoor to compute x_0 and x_1 in full.

At this point, the verifier randomly chooses to either request a projective measurement of the x register, ending the protocol, or to continue with the second and third rounds. In the former case, the prover communicates the result of that measurement, yielding either x_0 or x_1, and the verifier checks that indeed $f(x) = y$. In the latter case, the protocol proceeds with the final two rounds.

The second round of interaction converts the many-qubit superposition $|\psi\rangle = |x_0\rangle_{\mathsf{x}} + |x_1\rangle_{\mathsf{x}}$ into a single-qubit state $\{|0\rangle_{\mathsf{b}}, |1\rangle_{\mathsf{b}}, |+\rangle_{\mathsf{b}}, |-\rangle_{\mathsf{b}}\}$ on an ancilla qubit b. The final state of b depends on the values of both x_0 and x_1. The round begins with the verifier choosing a random bitstring r of the same length as x_0 and x_1, and sending it to the prover. Using a series of CNOT gates from the x register to b, the prover computes the state $|r \cdot x_0\rangle_{\mathsf{b}} |x_0\rangle_{\mathsf{x}} + |r \cdot x_1\rangle_{\mathsf{b}} |x_1\rangle_{\mathsf{x}}$, where $r \cdot x$ denotes the binary inner product. Finally, the prover measures the x register in the Hadamard basis, storing the result as a bitstring d which is sent to the verifier. This measurement disentangles x from b without collapsing b's superposition. At the end of the second round, the prover's state is $(-1)^{d \cdot x_0} |r \cdot x_0\rangle_{\mathsf{b}} + (-1)^{d \cdot x_1} |r \cdot x_1\rangle_{\mathsf{b}}$, which is one of $\{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$. Crucially, it is cryptographically hard to predict whether this state is one of $\{|0\rangle, |1\rangle\}$ or $\{|+\rangle, |-\rangle\}$.

The final round of our protocol can be understood in analogy to the CHSH game [128]. While the prover cannot extract the polarization axis from their single qubit (echoing the no-signaling property of CHSH), they can make a measurement that is *correlated* with it. This measurement outcome ultimately constitutes the proof of quantumness. In particular, the verifier requests a measurement in an intermediate basis, rotated from the Z axis around Y, by either $\theta = \pi/4$ or $-\pi/4$. Because the measurement basis is never perpendicular to the state, there will always be one outcome that is more likely than the other (specifically, with probability $\cos^2(\pi/8) \approx 0.85$). The verifier returns Accept if this "more likely" outcome is the one received.

In the next section, we prove that a quantum device can cause the verifier to Accept with substantially higher probability than any classical prover. A full test of quantum advantage would consist of running the protocol many times, until it can be established with high statistical confidence that the device has exceeded the classical probability bound.

5.3.2 Completeness and soundness

We now prove completeness (the noise-free quantum success probability) and soundness (an upper bound on the classical success probability). Recall that after the first round of the protocol, the verifier chooses to either request a standard basis measurement of the first register, or to continue with the second and third rounds. In the proofs below, we analyze

the prover's success probability across these two cases separately. We denote the probability that the verifier will accept the prover's string x in the first case as p_x, and the probability that the verifier will accept the single-qubit measurement result in the second case as p_{CHSH}.

5.3.2.1 Perfect quantum prover (completeness)

Theorem 2. *An error-free quantum device honestly following the interactive protocol will cause the verifier to return* Accept *with* $p_x = 1$ *and* $p_{\text{CHSH}} = \cos^2(\pi/8) \approx 0.85$.

Proof. If the verifier chooses to request a projective measurement of x after the first round, an honest quantum prover succeeds with probability $p_x = 1$ by inspection.

If the verifier chooses to instead perform the rest of the protocol, the prover will hold one of $\{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$ after round 2. In either measurement basis the verifier may request in round 3, there will be one outcome that occurs with probability $\cos^2(\pi/8)$, which is by construction the one the verifier accepts. Thus, an honest quantum prover has $p_{\text{CHSH}} = \cos^2(\pi/8) \approx 0.85$. □

5.3.2.2 Classical prover (soundness)

Theorem 3. *Assume the function family used in the interactive protocol is claw-free. Then,* p_x *and* p_{CHSH} *for any classical prover must obey the relation*

$$p_x + 4p_{\text{CHSH}} - 4 < \epsilon(n) \tag{5.1}$$

where ϵ *is a negligible function of* n*, the length of the function family's input strings.*

Proof. We prove by contradiction. Assume that there exists a classical machine $\mathcal{A}$ for which $p_x + 4p_{\text{CHSH}} - 4 \geq \mu(n)$, for a non-negligible function μ. We show that there exists another algorithm $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine to find a pair of colliding inputs to the claw-free function, a contradiction.

Given a claw-free function instance f_i, $\mathcal{B}$ acts as a simulated verifier for $\mathcal{A}$. $\mathcal{B}$ begins by supplying f_i to $\mathcal{A}$, after which $\mathcal{A}$ returns a value y, completing the first round of interaction. $\mathcal{B}$ now chooses to request the projective measurement of the x register, and stores the result as x_0. Letting p_{x_0} be the probability that x_0 is a valid preimage, by definition of p_x we have $p_{x_0} = p_x$.

Next, $\mathcal{B}$ *rewinds* the execution of $\mathcal{A}$, to its state before x_0 was requested. Crucially, rewinding is possible because $\mathcal{A}$ is a classical algorithm. $\mathcal{B}$ now proceeds by running $\mathcal{A}$ through the second and third rounds of the protocol for many different values of the bitstring r (Fig. 1), rewinding each time.

We now show that, for r selected uniformly at random, $\mathcal{B}$ can extract the value of the inner product $r \cdot x_1$ with probability $p_{r \cdot x_1} \geq 1 - 2(1 - p_{\text{CHSH}})$. $\mathcal{B}$ begins by sending r to $\mathcal{A}$, and receiving the bitstring d. $\mathcal{B}$ then requests the measurement result in both the $\theta = \pi/4$ and $\theta = -\pi/4$ bases, by rewinding in between. Supposing that both the received values are "correct" (i.e. would be accepted by the real verifier), they uniquely determine

the single-qubit state $|\psi\rangle \in \{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$ that would be held by an honest quantum prover. This state reveals whether $r \cdot x_0 = r \cdot x_1$, and because $\mathcal{B}$ already holds x_0, $\mathcal{B}$ can compute $r \cdot x_1$. We may define the probability (taken over all randomness except the choice of θ) that the prover returns an accepting value in the cases $\theta = \pi/4$ and $\theta = -\pi/4$ as $p_{\pi/4}$ and $p_{-\pi/4}$ respectively. Then, via union bound, the probability that both are indeed correct is $p_{r \cdot x_1} \geq 1 - (1 - p_{\pi/4}) - (1 - p_{-\pi/4})$. Considering that $p_{\text{CHSH}} = (p_{\pi/4} + p_{-\pi/4})/2$, we have $p_{r \cdot x_1} \geq 1 - 2(1 - p_{\text{CHSH}})$.

Now, we show that extracting $r \cdot x_1$ in this way allows x_1 to be determined in full even in the presence of noise, by rewinding many times and querying for specific (correlated) choices of r. In particular, the above construction is a noisy oracle to the encoding of x_1 under the Hadamard code. By the Goldreich-Levin theorem [149], list decoding applied to such an oracle will generate a polynomial-length list of candidates for x_1. If the noise rate of the oracle is noticeably less than $1/2$, x_1 will be contained in that list; $\mathcal{B}$ can iterate through the candidates until it finds one for which $f(x_1) = y$.

By Lemma 1 in the Methods, for a particular iteration of the protocol, the probability that list decoding succeeds is bounded by $p_{x_1} > 2p_{r \cdot x_1} - 1 - 2\mu'(n)$, for a noticeable function $\mu'(n)$ of our choice [5]. Setting $\mu'(n) = \mu(n)/4$ and combining with the previous result yields $p_{x_1} > 1 - 4(1 - p_{\text{CHSH}}) - \mu(n)/2$.

Finally, via union bound, the probability that $\mathcal{B}$ returns a claw is

$$P_{\mathcal{B}} \geq 1 - (1 - p_{x_0}) - (1 - p_{x_1}) > p_x + 4p_{\text{CHSH}} - 4 - \mu(n)/2$$

and via the assumption that $p_x + 4p_{\text{CHSH}} - 4 > \mu(n)$ we have

$$P_{\mathcal{B}} > \mu(n)/2$$

a contradiction. □

If we let $p_x = 1$, the bound requires that $p_{\text{CHSH}} < 3/4 + \epsilon(n)$ for a classical device, while $p_{\text{CHSH}} \approx 0.85$ for a quantum device, matching the classical and quantum success probabilities of the CHSH game. In Section 5.7.7, we provide an example of a classical algorithm saturating the bound with $p_x = 1$ and $p_{\text{CHSH}} = 3/4$.

5.3.3 Variations on the protocol

In this section we describe two variations on the protocol, the goal of both of which is to remove the need for the "preimage" test (Step 6a of Fig. 5.1). The main benefit of doing so is that it simplifies and improves the classical bound, to simply $p \leq 3/4 + \epsilon(n)$, where p now is the *overall* probability that the prover succeeds (equivalent to p_{CHSH} in the normal

[5] The oracle's noise rate is *not* simply $p_{r \cdot x_1}$: that is the probability that any single value $r \cdot x_1$ is correct, but all of the queries to the oracle are correlated (they are for the same iteration of the protocol, and thus the same value of y).

protocol, because p_x no longer exists). A secondary benefit is that it slightly simplifies the experimental implementation by making the protocol less complicated.

The idea is simple: in Step 6b of Fig. 5.1, instead of choosing a single random bitstring r, the verifier chooses two, r_0 and r_1. Then, in Step 7b, instead of using the single r for both x_0 and x_1, the prover instead computes $|r_0 \cdot x_0\rangle_{\mathsf{b}} |x_0\rangle_{\mathsf{x}} + |r_1 \cdot x_1\rangle_{\mathsf{b}} |x_1\rangle_{\mathsf{x}}$—a different inner product for each of the preimages. Applying the proof of Theorem 3 to this scheme, the responses of the classical machine $\mathcal{A}$ can be used to reconstruct whether $r_0 \cdot x_0 = r_1 \cdot x_1$ (where originally we reconstructed simply whether $r \cdot x_0 = r \cdot x_1$). The key insight is that the truth value of this new equality is equal to $(r_0||r_1) \cdot (x_0||x_1)$, where $||$ denote concatenation. This fact can be used to construct a noisy oracle for the inner product of $x_0||x_1$ with arbitrary strings, to which the Goldreich-Levin theorem can be applied to find $x_0||x_1$, fully revealing both x_0 and x_1. (This should be compared to the original proof, which could only decode $x_0 \oplus x_1$ via the Goldreich-Levin theorem, and thus required the preimage test to supply x_0 or x_1 and thus reveal the claw). Since x_0 and x_1 can both be reconstructed from only the CHSH portion of the protocol, the "preimage" test is not necessary for classical hardness and can be removed.

The downside of this variation of the protocol is that the prover needs to somehow be able to distinguish x_0 from x_1, so that the appropriate inner product can be taken with each. For many TCFs, such as the one based on LWE [39] and the DDH-based TCF we present in this chapter, this is not a problem—there is an extra qubit in the preimages which is in the state $|0\rangle$ for x_0 and $|1\rangle$ for x_1. However for $x^2 \bmod N$, it is not so straightforward. Via the Jacobi symbol it is technically possible to distinguish the two preimages, because it is a fact of $x^2 \bmod N$ that one preimage will have Jacobi symbol $+1$ and the other -1. However actually computing the Jacobi symbol is very expensive, much moreso than computing $x^2 \bmod N$ itself, defeating our goal of having an efficient implementation! Another somewhat less expensive strategy is to switch to the pair of functions $f_0(x) = x^2 \bmod N$ and $f_1(x) = 4x^2 \bmod N$, with their domain defined as the set of quadratic residues less than N (instead of the set of integers $[0, N/2]$ that were used before). By splitting into two functions we get the desired "marker" qubit distinguishing the two preimages, but we run into the problem of generating a uniform superposition of quadratic residues modulo N. To our knowledge the best way to generate such a superposition is to start with the set of all integers less than N, and square them. Then, another square must be taken to actually implement the TCF. So, it seems that using this TCF would require a quantum circuit twice as large as the original protocol using $x^2 \bmod N$—a tradeoff that is probably not worth it for the extra simplicity of removing the preimage test. That being said, if a function other than $x^2 \bmod N$ is used which does have the extra qubit, this variation is almost certainly the right choice.

We also note that we learned via personal correspondence with Eitan Porat, Zvika Brakerski, and Thomas Vidick that they found that the *original* protocol is actually classically hard without the preimage test. The intuitive idea is that we can learn more information from the "measurement results" than just whether $r \cdot x_0 = r \cdot x_1$. In particular, when that equality holds, we also get access to the *value* of $r \cdot x_0$ (and $r \cdot x_1$, since they are equal). With this extra information it is possible to use a more complicated scheme based on the Goldreich-Levin theorem to decode x_0 and x_1 in full, proving the hardness of passing just the

CHSH portion directly from the hardness of finding claws. However, apparently the classical bound is not quite as powerful due to the more complicated decoding process.

5.3.4 Robustness: Error mitigation via postselection

The existence of a finite gap between the classical and quantum success probabilities implies that our protocol can tolerate a certain amount of noise. A direct implementation of our interactive protocol on a noisy quantum device would require an overall fidelity of $\sim 83\%$ in order to exceed the classical bound [6]. To allow devices with lower fidelities to demonstrate quantum advantage, our protocol allows for a natural tradeoff between fidelity and runtime, such that the classical bound can, in principle, be exceeded with only a small [e.g. $1/\text{poly}(n)$] amount of coherence in the quantum device [7].

The key idea is based upon postselection. For most TCFs, there are many bitstrings of the correct length that are not valid outputs of f. Thus, if the prover detects such a y value in step 3 (Fig. 1), they can simply discard it and try again [8]. In principle, the verifier can even use their trapdoor data to silently detect and discard iterations of the protocol with invalid y [9]. Since y is a function of x_0 and x_1, one might hope that this postselection scheme also rejects states where x_0 or x_1 has become corrupt. Although this may not always be the case, we demonstrate numerically that this assumption holds for a specific implementation of $x^2 \bmod N$ in the following subsection. One could also compute a classical checksum of x_0 and x_1 before and after the main circuit to ensure that they have not changed during its execution. Assuming that such bit-flip errors are indeed rejected, the possibility remains of an error in the phase between $|x_0\rangle$ and $|x_1\rangle$. In Section 5.7.9, we demonstrate that a prover holding the correct bitstrings but with an error in the phase can still saturate the classical bound; if the prover can avoid phase errors even a small fraction of the time, they will push past the classical threshold.

5.3.4.1 Numerical analysis of the postselection scheme for $x^2 \bmod N$

Focusing on the function $f(x) = x^2 \bmod N$, we now explicitly analyze the effectiveness of the postselection scheme. Let m be the length of the outputs of this function. In this case,

[6] This number comes from solving the classical bound (Equation 5.1) for circuit fidelity $\mathcal{F}$, with $p_x = \mathcal{F}$ and $p_{\text{CHSH}} = 1/2 + \mathcal{F}/2$.

[7] This is true even if the coherence is exponentially small in n. Of course, with arbitrarily low coherence the runtime may become excessively large such that quantum advantage cannot be demonstrated—the point is that regardless of runtime, the classical probability bound can be exceeded with a device that has arbitrarily low circuit fidelity.

[8] This scheme will only remove errors in the first round of the protocol, but fortunately, one expects the overwhelming majority of the quantum computation, and thus also the majority of errors, to occur in that round.

[9] This procedure does not leak data to a classical cheater, because the verifier does not communicate which runs were discarded. Furthermore, it does not affect the soundness of Theorem 3, because the machine $\mathcal{B}$ in that theorem's proof can simply iterate until it encounters a valid y.

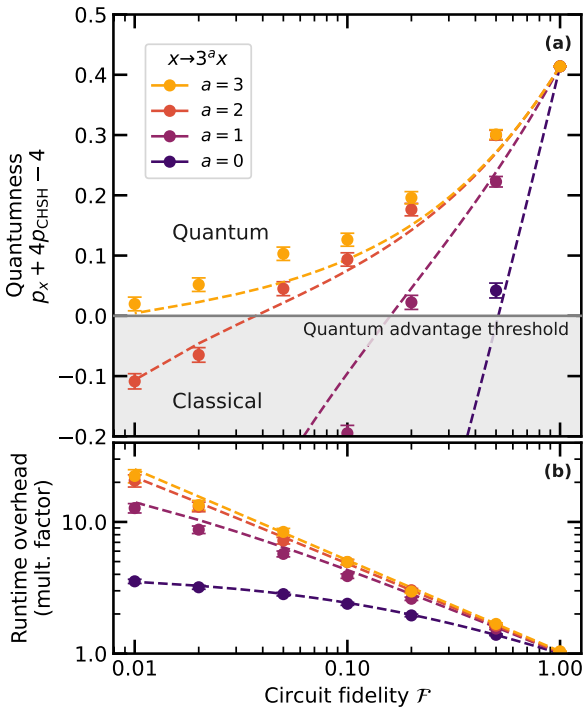


Figure 5.2: **Performance of our post-selection scheme.** Redundancy is added to the function x^2 mod N by mapping it to $(3^a x)^2$ mod $3^{2a}N$. Numerical simulations are performed on a quantum circuit implementing the Karatsuba algorithm for $a = \{0, 1, 2, 3\}$ (see Section 5.7.9). **(a)** "Quantumness" measured in terms of the classical bound from Eqn. 5.1 as a function of the total circuit fidelity. With $a = 3$, even a quantum device with only 1% circuit fidelity can demonstrate quantum advantage. **(b)** Depicts the increased runtime associated the post-selection scheme, which arises from a combination of slightly larger circuit sizes and the need to re-run the circuit multiple times. The latter is by far the dominant effect. Dashed lines are a theory prediction with no fit parameters; points are the result of numerical simulations at $n = 512$ bits and error bars depict 2σ uncertainty.

approximately 1/4 of the bitstrings of length m are valid outputs, so one would naively expect to reject about 3/4 of corrupted bitstrings. By introducing additional redundancy into the outputs of f and thus increasing m, one can further decrease the probability that a corrupted y will incorrectly be accepted. As an example, let us consider mapping $x^2 \bmod N$ to the function $(kx)^2 \bmod k^2N$ for some integer k. This is particularly convenient because the prover can validate y by simply checking whether it is a multiple of k^2. Moreover, the mapping adds only $\log k$ bits to the size of the problem, while rejecting a fraction $1 - 1/k^2$ of corrupted bitstrings.

We perform extensive numerical simulations demonstrating that postselection allows for quantum advantage to be achieved using noisy devices with low circuit fidelities (Fig. 2). We simulate quantum circuits for $(kx)^2 \bmod k^2N$ at a problem size of $n = 512$ bits. Assuming a uniform gate fidelity across the circuit, we analyze the success rate of a quantum prover for $k = 3^a$ and $a = \{0, 1, 2, 3\}$. For these simulations we use our implementation of the Karatsuba algorithm (see Section 5.5.1) because it is the most efficient in terms of gate count and depth. The choice of $k = 3^a$, and details of the simulation, are explained in Section 5.7.9.

For $a = 0$, the circuit implements our original function $x^2 \bmod N$, where in the absence of postselection, an overall circuit fidelity of $\mathcal{F} \sim 0.83$ is required to achieve quantum advantage. As depicted in Fig. 5.2(a), even for $a = 0$, our postselection scheme improves the advantage threshold down to $\mathcal{F} \sim 0.51$. For $a = 2$, circuit fidelities with $\mathcal{F} \gtrsim 0.1$ remain well above the quantum advantage threshold, while for $a = 3$ the required circuit fidelity drops below 1%.

However, there is a tradeoff. In particular, one expects the overall runtime to increase for two reasons: (i) there will be a slight increase in the circuit size for $a > 0$ and (ii) one may need to re-run the quantum circuit many times until a valid y is measured. Somewhat remarkably, a runtime overhead of only 4.7x already enables quantum advantage to be achieved with an overall circuit fidelity of 10% [Fig. 5.2(b)]. Crucially, this increase in runtime is overwhelmingly due to re-running the quantum circuit and does not imply the need for longer experimental coherence times.

5.3.5 Efficient quantum evaluation of irreversible classical circuits

The central computational step in our interactive protocol (i.e. step 2, Fig. 5.1) is for the prover to apply a unitary of the form:

$$\mathcal{U}_{f_i} \sum_x |x\rangle_{\mathsf{x}} |0^{\otimes m}\rangle_{\mathsf{y}} = \sum_x |x\rangle_{\mathsf{x}} |f_i(x)\rangle_{\mathsf{y}}\,, \tag{5.2}$$

where $f_i(x)$ is a classical function and m is the length of the output register. This type of unitary operation is ubiquitous across quantum algorithms, and a common strategy for its implementation is to convert the gates of a classical circuit into quantum gates. Generically, this process induces substantial overhead in both time and space complexity owing to the need to make the circuit reversible to preserve unitarity [137], [138]. This reversibility is often achieved by using an additional register, g, of so-called "garbage bits" and implementing:

$\mathcal{U}'_{f_i} \sum_x |x\rangle_{\mathsf{x}} |0^{\otimes m}\rangle_{\mathsf{y}} |0^{\otimes l}\rangle_{\mathsf{g}} = \sum_x |x\rangle_{\mathsf{x}} |f_i(x)\rangle_{\mathsf{y}} |g_i(x)\rangle_{\mathsf{g}}$. For each gate in the classical circuit, enough garbage bits are added to make the operation injective. In general, to maintain coherence, these bits cannot be discarded but must be "uncomputed" later, adding significant complexity to the circuits.

A particularly appealing feature of our protocol is the existence of a measurement scheme to discard garbage bits, allowing for the direct mapping of classical to quantum circuits with no overhead. Specifically, we envision the prover measuring the qubits of the g register in the Hadamard basis and storing the results as a bitstring h, yielding the state,

$$|\psi\rangle = \sum_x (-1)^{h \cdot g_i(x)} |x\rangle_{\mathsf{x}} |f_i(x)\rangle_{\mathsf{y}} . \tag{5.3}$$

The prover has avoided the need to do any uncomputation of the garbage bits, at the expense of introducing phase flips onto some elements of the superposition. These phase flips do not affect the protocol, so long as the verifier can determine them. While classically computing $h \cdot g_i(x)$ is efficient for any x, computing it for *all* terms in the superposition is infeasible for the verifier. However, our protocol provides a natural way around this. The verifier can wait until the prover has collapsed the superposition onto x_0 and x_1, before evaluating $g_i(x)$ only on those two inputs [10].

Crucially, the prover can measure away garbage qubits as soon as they would be discarded classically, instead of waiting until the computation has completed. If these qubits are then reused, the quantum circuit will use no more space than the classical one. This feature allows for significant improvements in both gate depth and qubit number for practical implementations of the protocol (see last rows of Table I in Methods). We note that performing many individual measurements on a subset of the qubits is difficult on some experimental systems, which may make this technique challenging to use in practice. However, recent hardware advances have demonstrated these "intermediate measurements" in practice with high fidelity, for example by spatially shuttling trapped ions [150], [151]. We thus expect that the capability to perform partial measurements will not be a barrier in the near term. This issue can also be mitigated somewhat by collecting ancilla qubits and measuring them in batches rather than one-by-one, allowing for a direct trade-off between ancilla usage and the number of partial measurements.

5.4 The search for alternative trapdoor claw-free functions

Before moving on to proposals for the physical implementation of this protocol, I would like to briefly summarize some of my *unsuccessful* efforts to find new constructions for trapdoor claw-free functions, in hope that it can be helpful for anyone trying to do so in the future.

[10] This is true because $g_i(x)$ is the result of adding extra output bits to the gates of a classical circuit, which is efficient to evaluate on any input.

Broadly, the goal is to come up with a TCF that can be implemented in as small a quantum circuit as possible—primarily in terms of number of qubits and number of gates. Other potentially important statistics include circuit depth (parallelism) and spatial locality of the gates.

We will focus on the x^2 mod N-based TCF in the later sections of this chapter because it seems to strike the best balance in achieving the goals above, but it is not perfect because the modulus N needs to be quite large for the problem to be classically hard—which has negative consequences for both the qubit and gate counts. For example, considering just qubit count for the moment, if we desire the security of a 1024-bit modulus, there is a hard lower bound of 1024 qubits required to implement the circuit (and in practice, the circuit will probably require a considerable amount more than that). This should be compared to the fact that in the average case, circuits of fewer than 100 qubits with sufficient depth are infeasible to classically simulate—so there is a large gap between the hardness of *simulation* and the hardness of the *cryptography*. Ideally, we would make that gap as small as possible. The DDH-based TCF also proposed in this chapter has the potential to improve the gap considerably: when implemented using elliptic curve cryptography, the group elements can be as small as a couple hundred bits long and the hardness assumption remains secure. Unfortunately, the gate count required to implement that TCF is *dramatically* worse than for x^2 mod N, and that is why we do not focus our efforts on building circuits for it.

Given these considerations, I expended a considerable effort in looking for other cryptographic assumptions that could be used to build a trapdoor claw-free function. Coming up with new, more efficient TCFs directly from the ground up is a daunting pursuit: finding ways to make public-key cryptography more efficient is of central concern for *classical* cryptography, so it has been a subject of intense research for years. So instead of trying to break new ground there, a more modest goal is to take other existing schemes for public-key cryptography which do not have the precise structure of a TCF, and build TCFs out of them.

In my efforts to do so, one promising candidate seemed to be the Learning Parity with Noise (LPN) problem, which has found use for classical cryptography in devices with very limited computational power such as RFID cards. The structure of the LPN problem is similar to that of LWE, but the linear algebra takes place over the field F_2 of binary numbers instead of integers modulo some large q. [152] To be explicit, consider a binary matrix $A \in \{0,1\}^{m \times n}$, with, say, $m = 2n$. For a secret string $s \in \{0,1\}^n$ and "error" vector $e \in \{0,1\}^m$, consider the "noisy" image of s defined as $y = As + e$. The LPN hardness assumption states that for appropriate setting of the problem parameters, given only y and A it is computationally hard (even for a quantum computer) to recover s unless A has some special structure.[11] Obviously this is the case if the noise vector e is overwhelming; the problem is interesting because this seems to hold even when e is quite sparse (most entries

[11] When I first learned about LPN I got extremely interested in exploring the classical hardness of the problem. I wrote the first (to my knowledge) GPU-accelerated solver for it, and ended up breaking the world record for the largest instance that had been solved. After about a year I was unseated by another GPU-based implementation. The competition can be found here: `https://decodingchallenge.org/syndrome`, I encourage the reader to try their hand at it!

are zero). One can see the potential here for simplicity of implementation: performing the linear algebra requires only addition and multiplication of numbers in F_2, which corresponds simply to XOR and AND gates. This is dramatically less complicated than the addition and multiplication circuits for integers modulo some large value q, which are required to implement LWE.

The challenge is to figure out how to build a TCF out of this hardness assumption. Considering the similarity of the LWE and LPN problems, an obvious idea is to follow the structure of the LWE TCF, and define two functions roughly as

$$f_0(x) = Ax \tag{5.4}$$

$$f_1(x) = Ax + y \tag{5.5}$$

Using the definition of y, we see that $f_1(x) = A(x + s) + e$, and thus that for a pair (x_0, x_1) where $x_0 = x_1 + s$, we have $f_0(x_0) = f_1(x_1) + e$—that is, it is *almost* a claw, aside from the error vector e (which has most entries set to zero). But for the protocol to work, we need an exact collision, rather than an approximate one. In LWE, this is done by adding *extra* error e' to the output of both f_0 and f_1, to "smear out" the values. If the distribution of e' is sufficiently wider than the distribution of e, then e disappears into the noise and the probability distributions have good overlap, yielding collisions. Unfortunately, despite considerable effort, it does not seem that it is possible to do the same trick with LPN. The problem stems from the same reason that LPN seemed promising: the linear algebra is over F_2 instead of F_q. Intuitively, because each value can only be 0 or 1, there is simply no "room" to have a wider probability distribution for the elements of an extra noise vector e'. (In fact, the LWE TCF requires q to be very large precisely for this reason). Perhaps there is some other scheme to create exact collisions from these near-collisions in LPN, like rounding the outputs somehow, but I was never able to find one.

Looking at the problem more broadly, it actually seems very unlikely that it is possible to create perfect collisions in this way, because it turns out doing so would break the assumption of post-quantum hardness of LPN, which is widely believed to hold. The reason is because this pair of functions could be used as an oracle for Simon's algorithm, which would allow a quantum device to very efficiently find s. [153] The only hope seems to be the fact that Simon's algorithm requires the functions to perfectly collide all but an exponentially small fraction of the time, so perhaps if the collisions are not perfect, the LPN assumption would not be broken. However, even broadening the search to look for such "noisy" TCFs based on LPN has yet to yield any useful constructions. One last idea is that maybe there is a way to use LPN in an entirely different manner to create a TCF—but for that, it's not even clear where to start.

5.5 Quantum circuits for trapdoor claw-free functions

As just discussed, while all of the trapdoor, claw-free functions listed in Table 5.1 can be utilized within our interactive protocol, each has its own set of advantages and disadvantages.

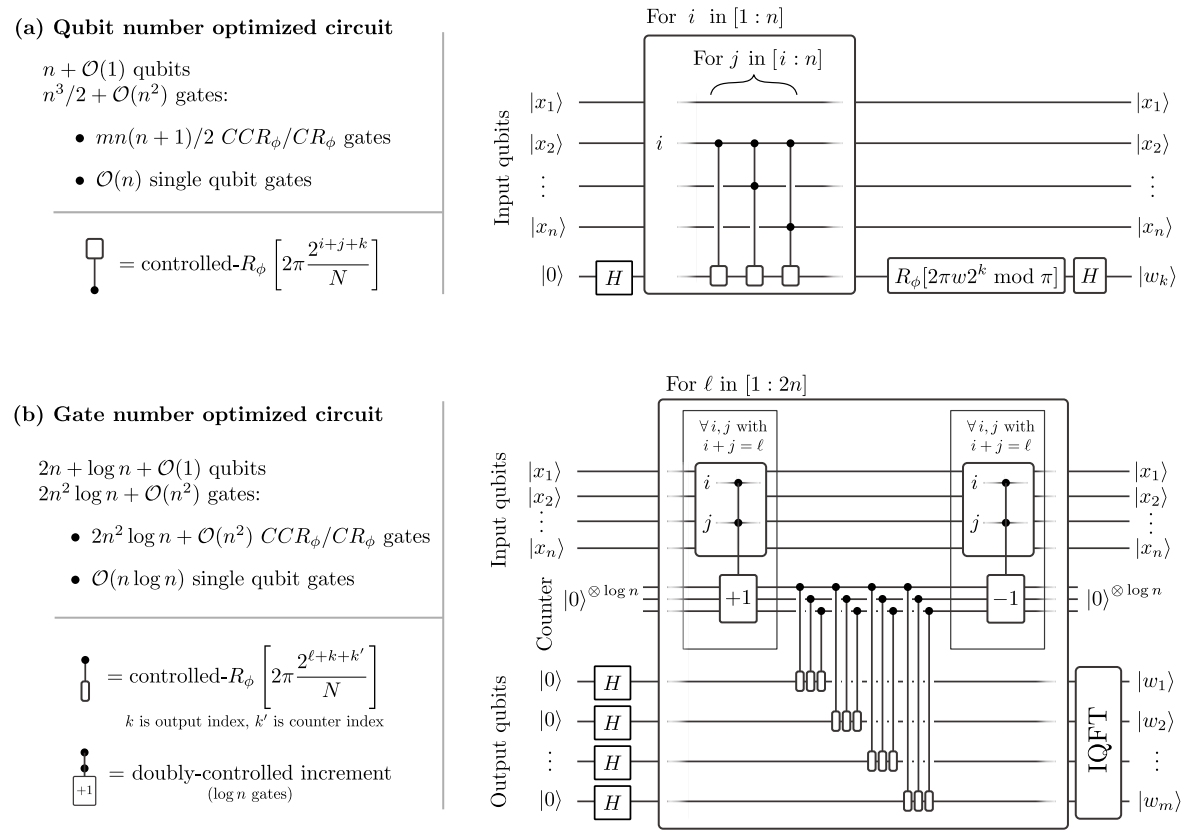


Figure 5.3: **Quantum circuits implementing step 2 of our interactive protocol for $f(x) = x^2 \bmod N$.** n is the length of the input register, and $m = n + \mathcal{O}(1)$ is the length of the output register. **(a)** Depicts a quantum circuit optimized for qubit number. The circuit shown computes the k^{th} bit of $w = x^2/N$ and should be iterated for k. This iteration should begin at the least significant bit to ensure that the final phase rotation can be estimated classically. Note that the only entangling operations necessary for the circuit are doubly-controlled gates, which can be natively implemented using the Rydberg blockade (see Section 5.5.3). **(b)** Depicts a quantum circuit optimized for gate number. By combining gates of equal phase, one can reduce the overall circuit complexity to $\mathcal{O}(n^2 \log n)$ gates. We note that neither circuit requires use of the "garbage bit" procedure described in Section 5.3.5; this design choice reduces measurement complexity. If desired, that procedure could be applied to the counter register of circuit (b) in place of the controlled-decrement operation.

For example, the TCF based on the Diffie-Hellman problem (described in the Methods) already enables a demonstration of quantum advantage at a key size of 160 bits (with a hardness equivalent to 1024 bit integer factorization [136]); however, building a circuit for this TCF requires a quantum implementation of Euclid's algorithm, which is challenging [154]. Thus, we focus on designing quantum circuits implementing Rabin's function, $x^2 \bmod N$.

5.5.1 Quantum circuits for $x^2 \bmod N$

In Chapter 7 we present what to our knowledge are the most highly optimized circuits known for $x^2 \bmod N$. Here, we present four more basic circuits, that exhibit the range of possible implementations of $x^2 \bmod N$ and provide a good comparison for the optimizations in that chapter. For the circuits presented here, implementations in `Python` using the `Cirq` library are included as supplementary files [12]. The first two are quantum implementations of classical circuits for the Karatsuba and "schoolbook" classical integer multiplication algorithms, where we leverage the reversibility optimizations described in Section 5.3.5 (see Section 5.7.8 for details of their implementation). The latter pair, which we call the "phase circuits" and describe below, are intrinsically quantum algorithms that use Ising interactions to directly compute $x^2 \bmod N$ in the phase. Using those circuits, we propose a near-term demonstration of our interactive protocol on a Rydberg-based quantum computer [143], [146]; crucially, the so-called "Rydberg blockade" interaction *natively* realizes multi-qubit controlled phase rotations, from which the entire circuits shown in Figure 3 are built (up to single qubit rotations). A comparison of approximate gate counts for each of the four circuits can be seen in Table I in the Methods. Of the circuits explored here, the Karatsuba algorithm is the most efficient in total gates and circuit depth, while the phase circuits are most efficient in terms of qubit usage and measurement complexity. Chapter 7 manages to combine the benefits of both, yielding circuits with gate counts better than the Karatsuba circuits here and qubit usage and measurement complexity comparable to the phase circuits.

5.5.2 Phase circuits

We now describe the two circuits, amenable to near-term quantum devices, that utilize quantum phase estimation to implement the function $f(x) = x^2 \bmod N$. The intuition behind our approach is as follows: we will compute x^2/N in the phase and transfer it to an output register via an inverse quantum Fourier transform [156], [157]; the modulo operation occurs automatically as the phase wraps around the unit circle, avoiding the need for a separate reduction step.

In order to implement $\sum_x |x\rangle_{\mathsf{x}} |x^2 \bmod N\rangle_{\mathsf{y}}$, we design a circuit to compute:

$$(\mathbb{I} \otimes \mathrm{IQFT})\, \tilde{\mathcal{U}}_{w_N}\, (\mathbb{I} \otimes \mathrm{H}^{\otimes \mathrm{m}})\, |x\rangle\, |0^{\otimes m}\rangle = |x\rangle\, |w\rangle \tag{5.6}$$

[12] Code is available at `https://github.com/GregDMeyer/quantum-advantage` and is archived on Zenodo [155]

where H is a Hadamard gate, IQFT represents an inverse quantum Fourier transform, $w \equiv x^2/N = 0.w_1 w_2 \cdots w_m$ is an m-bit binary fraction [13], and $\tilde{\mathcal{U}}_{w_N}$ is the diagonal unitary,

$$\tilde{\mathcal{U}}_{w_N} |x\rangle |z\rangle = \exp\left(2\pi i \frac{x^2}{N} z\right) |x\rangle |z\rangle . \tag{5.7}$$

The simplest circuit to implement $\tilde{\mathcal{U}}_{w_N}$ simply decomposese x and z in binary, and performs a digit-by-digit multiplication using the schoolbook algorithm:

$$\exp\left(2\pi i \frac{x^2}{N} z\right) = \prod_{i,j,k} \exp\left(2\pi i \frac{2^{i+j+k}}{N} x_i x_j z_k\right), \tag{5.8}$$

With this, one immediately finds that $\tilde{\mathcal{U}}_{w_N}$ is equivalent to applying a series of controlled-controlled-phase rotation gates of angle,

$$\phi_{ijk} = \frac{2\pi 2^{i+j+k}}{N} \pmod{2\pi}. \tag{5.9}$$

Here, the control qubits are i, j in the x register, while the target qubit is k in the y register. Crucially, the value of this phase for any i, j, k can be computed classically when the circuit is compiled.

Figure 5.3 shows two explicit circuits to implement $\tilde{\mathcal{U}}_{w_N}$, one optimizing for qubit count, and the other optimizing for gate count. The first circuit [Fig. 5.3(a)] takes advantage of the fact that the output register is measured immediately after it is computed; this allows one to replace the m output qubits with a single qubit that is measured and reused m times. Moreover, by replacing groups of doubly-controlled gates with a Toffoli and a series of singly-controlled gates, one ultimately arrives at an implementation, which requires $n^3/2 + \mathcal{O}(n^2)$ gates, but only $n + \mathcal{O}(1)$ qubits. We note that this does require individual measurement and re-use of qubits, which has been a challenge for experiments; recent experiments however have demonstrated this capability [150], [151].

The second circuit [Fig. 5.3(b)], which optimizes for gate count, leverages the fact that ϕ_{ijk} (Eqn. 5.9) only depends on $i + j + k$, allowing one to combine gates with a common sum. In this case, one can define $\ell = i + j$ and then, for each value of ℓ, simply "count" the number of values of i, j for which both control qubits are 1. By then performing controlled gates off of the qubits of the counter register, one can reduce the total gate complexity by a factor of $n/\log n$, leading to a implementation with $2n^2 \log n + \mathcal{O}(n^2)$ gates.

5.5.3 Experimental implementation

Motivated by recent advances in the creation and control of many-body entanglement in programmable quantum systems [32], [158]–[160], we propose an experimental implementation of

[13] We must take $m > n + \mathcal{O}(1)$ to sufficiently resolve the value x^2 mod N in post-processing

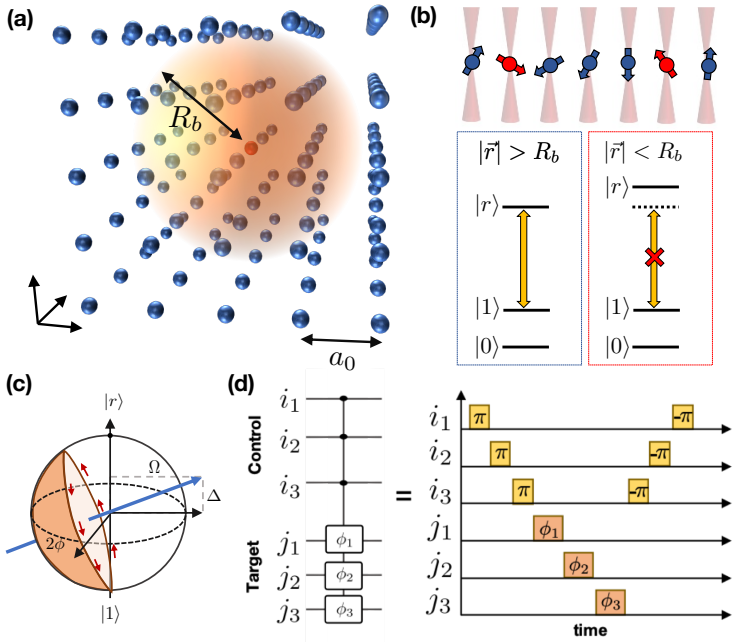


Figure 5.4: **Physical implementation in a Rydberg atom quantum computer. (a)** Schematic illustration of a three dimensional array of neutral atoms with Rydberg blockade interactions. The blockade radius can be significantly larger than the inter-atom spacing, enabling multi-qubit entangling operations. **(b)** As an example, Rydberg atoms can be trapped in an optical tweezer array. The presence of an atom in a Rydberg excited state (red) shifts the energy levels of nearby atoms (blue), preventing the driving field (yellow arrow) from exciting them to their Rydberg state, $|r\rangle$. **(c)** A single qubit phase rotation can be implemented by an off-resonant Rabi oscillation between one of the qubit states, e.g., $|1\rangle$, and the Rydberg excited state. This imprints a tunable, geometric phase ϕ, which is determined by the detuning Δ and Rabi frequency Ω. **(d)** Multi-qubit controlled-phase rotations are implemented via a sequence of π-pulses between the $|0\rangle \leftrightarrow |r\rangle$ transition of control atoms (yellow) and off-resonant Rabi oscillations on the target atoms (orange).

our interactive protocol based upon neutral atoms coupled to Rydberg states [146]. We envision a three dimensional system of either alkali or alkaline-earth atoms trapped in an optical lattice or optical tweezer array [Fig. 5.4(a)] [161]–[163]. To be specific, we consider ^{87}Rb with an effective qubit degree of freedom encoded in hyperfine states: $|0\rangle = |F = 1, m_F = 0\rangle$ and $|1\rangle = |F = 2, m_F = 0\rangle$. Gates between atoms are mediated by coupling to a highly-excited Rydberg state $|r\rangle$, whose large polarizability leads to strong van der Waals interactions. This microscopic interaction enables the so-called Rydberg "blockade" mechanism—when a single atom is driven to its Rydberg state, all other atoms within a blockade radius, R_b, become off-resonant from the drive, thereby suppressing their excitation [Fig. 5.4(a,b)] [142].

Somewhat remarkably, this blockade interaction enables the *native* implementation of all multi-qubit-controlled phase gates depicted in the circuits in Figure 5.3. In particular, con-

sider the goal of applying a $C^k R_\phi^\ell$ gate; this gate applies phase rotations, $\{\phi_1, \phi_2, \ldots, \phi_\ell\}$, to target qubits $\{j_1, j_2, \ldots j_\ell\}$ if all k control qubits $\{i_1, i_2, \ldots i_k\}$ are in the $|1\rangle$ state [Fig. 5.4(d)]. Experimentally, this can be implemented as follows: (i) sequentially apply (in any order) resonant π-pulses on the $|0\rangle \leftrightarrow |r\rangle$ transition for the k desired control atoms, (ii) off-resonantly drive the $|1\rangle \leftrightarrow |r\rangle$ transition of each target atom with detuning Δ and Rabi frequency Ω for a time duration $T = 2\pi/(\Omega^2 + \Delta^2)^{1/2}$ [Fig. 5.4(c)], (iii) sequentially apply [in the opposite order as in (i)] resonant $-\pi$-pulses (i.e. π-pulses with the opposite phase) to the k control atoms to bring them back to their original state. The intuition for why this experimental sequence implements the $C^k R_\phi^\ell$ gate is straightforward. The first step creates a blockade if any of the control qubits are in the $|0\rangle$ state, while the second step imprints a phase, $\phi = \pi(1 - \Delta/\sqrt{\Delta^2 + \Omega^2})$, on the $|1\rangle$ state, only in the absence of a blockade. Note that tuning the values of ϕ_i for each of the target qubits simply corresponds to adjusting the detuning and Rabi frequency of the off-resonant drive in the second step [Fig. 5.4(c,d)].

Demonstrations of our protocol can already be implemented in current generation Rydberg experiments, where a number of essential features have recently been shown, including: 1) the coherent manipulation of individual qubits trapped in a 3D tweezer array [161], [162], 2) the deterministic loading of atoms in a 3D optical lattice [163], and 3) fast entangling gate operations with fidelities, $F \geq 0.974$ [143]–[145]. In order to estimate the number of entangling gates achievable within decoherence time scales, let us imagine choosing a Rydberg state with a principal quantum number $n \approx 70$. This yields a strong van der Waals interaction, $V(\vec{r}) = C_6/r^6$, with a C_6 coefficient $\sim (2\pi)\, 880$ GHz$\cdot\mu$m^6 [164]. Combined with a coherent driving field of Rabi frequency $\Omega \sim (2\pi)\, 1 - 10$ MHz, the van der Waals interaction can lead to a blockade radius of up to, $R_b = (C_6/\Omega)^{1/6} \sim 10\,\mu$m. Within this radius, one can arrange $\sim 10^2$ all-to-all interacting qubits, assuming an atom-to-atom spacing of approximately, $a_0 \approx 2\mu$m [14]. In current experiments, the decoherence associated with the Rydberg transition is typically limited by a combination of inhomogeneous Doppler shifts and laser phase/intensity noise, leading to $1/T_2 \sim 10 - 100$ kHz [143], [165], [166]. Taking everything together, one should be able to perform $\sim 10^3$ entangling gates before decoherence occurs (this is comparable to the number of two-qubit entangling gates possible in other state-of-the-art platforms [32], [167]). While this falls short of enabling an immediate full-scale demonstration of classically verifiable quantum advantage, we hasten to emphasize that the ability to directly perform multi-qubit entangling operations significantly reduces the cost of implementing our interactive protocol. For example, the standard decomposition of a Toffoli gate uses 6 CNOT gates and 7 T and $T^\dagger$ gates, with a gate depth of 12 [168]–[170]; an equivalent three qubit gate can be performed in a single step via the Rydberg blockade mechanism.

[14] We note that this spacing is ultimately limited by a combination of the optical diffraction limit and the orbital size of $n \approx 70$ Rydberg states.

5.6 Conclusion and outlook

The interplay between classical and quantum complexities ultimately determines the threshold for any quantum advantage scheme. Here, we have proposed a novel interactive protocol for classically verifiable quantum advantage based upon trapdoor claw-free functions; in addition to proposing two new TCFs [Table 5.1], we also provide explicit quantum circuits that leverage the microscopic interactions present in a Rydberg-based quantum computer. Our work allows near-term quantum devices to move one step closer toward a loophole-free demonstration of quantum advantage and also has opened the door to a number of promising future directions.

First, the proof of soundness contained in this chapter only applies to classical adversaries. Since the work in this chapter was originally published, a work by several colleagues and myself has extended the cryptographic proofs to the quantum case. In particular, we show that when the protocol from this work is instantiated with a quantum secure TCF like the one based off of LWE, it can be used to certify certain facts about the inner workings of the quantum device, with implications for quantum cryptographic applications such as certifiable random number generation or even the verification of arbitrary computations. [49] Second, our work has motivated the search for new trapdoor claw-free functions, as discussed in Section 5.4. At least one new construction has been discovered since this work was published; ideally more will be found as the search continues. [40] More broadly, one could also attempt to build modified protocols, which simplify either the requirements on the cryptographic function or the interactions; interestingly, recent work has demonstrated that using random oracles can remove the need for interactions in a TCF-based proof of quantumness [41], or even remove the need for a TCF entirely! [44] Finally, while we have focused our experimental discussions on Rydberg atoms, a number of other platforms also exhibit features that facilitate the protocol's implementation. For example, both trapped ions and cavity-QED systems can allow all-to-all connectivity, while superconducting qubits can be engineered to have biased noise [171]. This latter feature would allow noise to be concentrated into error modes detectable by our proposed post-selection scheme.

5.7 Additional proofs and data

5.7.1 List decoding lemma

In this section we prove a bound on the probability that list decoding will succeed for a particular value of y, given an oracle's noise rate over *all* values of y. Recall that by the Goldreich-Levin theorem [149], list decoding of the Hadamard code is possible if the noise rate is noticeably less than $1/2$.

Lemma 1. *Consider a binary-valued function over two inputs $g : Y \times \{0,1\}^n \to \{0,1\}$, and a noisy oracle $\mathcal{G}$ to that function. Assuming some distribution of values $y \in Y$ and $r \in \{0,1\}^n$, define $\epsilon \equiv \Pr_{y,r}[\mathcal{G}(y,r) \neq g(y,r)]$ as the "noise rate" of the oracle. Now define*

the conditional noise rate for a particular $y \in Y$ as

$$\epsilon_y \equiv \Pr_r \left[\mathcal{G}(y,r) \neq g(y,r)\right] \tag{5.10}$$

Then, the probability that ϵ_y is less than $1/2-\mu(n)$ for any positive function μ, over randomly selected y, is

$$p_{\text{good}} \equiv \Pr_y \left[\epsilon_y < 1/2 - \mu(n)\right] \geq 1 - 2\epsilon - 2\mu(n). \tag{5.11}$$

Proof. Let $S \subseteq Y$ be the set of y values for which $\epsilon_y < 1/2 - \mu(n)$. Then by definition we have

$$\epsilon = p_{\text{good}} \cdot \epsilon_{y\in S} + (1 - p_{\text{good}}) \cdot \epsilon_{y\notin S} \tag{5.12}$$

Noting that we must have $\epsilon_y \geq 1/2 - \mu(n)$ for $y \notin S$ by definition, we may minimize the right hand side of Equation 5.12, yielding the bound

$$\epsilon > p_{\text{good}} \cdot 0 + (1 - p_{\text{good}}) \cdot (1/2 - \mu(n)) \tag{5.13}$$

Rearranging this expression we arrive at

$$p_{\text{good}} > 1 - 2\epsilon - 2\mu(n)$$

which is what we desired to show. □

5.7.2 Trapdoor claw-free function constructions

Here we present two trapdoor claw-free function families (TCFs) for use in the protocol of this paper. These families are defined by three algorithms: Gen, a probabilistic algorithm which selects an index i specifying one function in the family and outputs the corresponding trapdoor data t; f_i, the definition of the function itself; and T, a trapdoor algorithm which efficiently inverts f_i for any i, given the corresponding trapdoor data t. Here we provide the definitions of the function families; proofs of their cryptographic properties are included in the supplementary information. In these definitions we use a security parameter λ following the notation of cryptographic literature; λ is informally equivalent to the "problem size" n defined in the main text as the length of the TCF input string.

5.7.2.1 TCF from Rabin's function $x^2 \bmod N$

"Rabin's function" $f_N(x) = x^2 \bmod N$, with N the product of two primes, was first used in the context of public-key cryptography and digital signatures [132], [133]. We use it to define the trapdoor claw-free function family $\mathcal{F}_{\text{Rabin}}$, as follows.

Function generation

$\mathsf{Gen}\left(1^\lambda\right)$

1. Randomly choose two prime numbers p and q of length $\lambda/2$ bits, with $p \bmod 4 \equiv q \bmod 4 \equiv 3 \bmod 4$ [15].

2. Return $N = pq$ as the function index, and the tuple (p, q) as the trapdoor data.

Function definition

$f_N : [N/2] \to [N]$ is defined as

$$f_N(x) = x^2 \bmod N \tag{5.14}$$

The domain is restricted to $[N/2]$ to remove extra trivial collisions of the form $(x, -x)$.

Trapdoor

The trapdoor algorithm is the same as the decryption algorithm of the Rabin cryptosystem [132]. On input y and key (p, q), the Rabin decryption algorithm returns four integers $(x_0, x_1, -x_0, -x_1)$ in the range [0, N). x_0 and x_1 can then be selected by choosing the two values that are smaller than $N/2$. See proof in supplementary information for an overview of the algorithm.

5.7.2.2 TCF from Decisional Diffie-Hellman

We now present a trapdoor claw-free function family $\mathcal{F}_{\text{DDH}}$ based on the decisional Diffie-Hellman problem (DDH). DDH is defined for a multiplicative group $\mathbb{G}$; informally, the DDH assumption states that for a group generator g and two integers a and b, given g, g^a, and g^b it is computationally hard to distinguish g^{ab} from a random group element. We expand on a known DDH-based trapdoor one-way function construction [130], [131], adding the claw-free property to construct a TCF.

Function generation

$\mathsf{Gen}\left(1^\lambda\right)$

1. Choose a group $\mathbb{G}$ of order $q \sim \mathcal{O}\left(2^\lambda\right)$, and a generator g for that group.

2. For dimension $k > \log_2 q$ choose a random invertible matrix $\mathbf{M} \in \mathbb{Z}_q^{k \times k}$.

3. Compute $g^{\mathbf{M}} = \left(g^{\mathbf{M}_{ij}}\right) \in \mathbb{G}^{k \times k}$ (element-wise exponentiation).

4. Choose a secret vector $\mathbf{s} \in \{0, 1\}^k$; compute the vector $g^{\mathbf{Ms}}$ (where $\mathbf{Ms}$ is the matrix-vector product, and again the exponentiation is element-wise).

[15] In practice, p and q must be selected with some care such that Fermat factorization and Pollard's $p - 1$ algorithm [172] cannot be used to efficiently factor N classically. Selecting p and q in the same manner as for RSA encryption would be effective [173].

5. Publish the pair $\left(g^{\mathbf{M}}, g^{\mathbf{Ms}}\right)$, retain $(g, \mathbf{M}, \mathbf{s})$ as the trapdoor data.

Function definition

Let d be a power of two with $d \sim \mathcal{O}\left(k^2\right)$. We define the function f_i as $f_i(b||\mathbf{x}) := f_{i,b}(\mathbf{x})$, where $||$ denotes concatenation, for a pair of functions $f_{i,b} : \mathbb{Z}_d^k \to \mathbb{G}^k$:

$$f_{i,0}\left(\mathbf{x}\right) = g^{\mathbf{Mx}} \tag{5.15}$$

$$f_{i,1}\left(\mathbf{x}\right) = g^{\mathbf{Mx}} g^{\mathbf{Ms}} = g^{\mathbf{M(x+s)}} \tag{5.16}$$

Trapdoor

The algorithm takes as input the trapdoor data $(g, \mathbf{M}, \mathbf{s})$ and a value $y = g^{\mathbf{Mx}_0} = g^{\mathbf{M}(\mathbf{x}_1+\mathbf{s})}$, and returns the claw $(\mathbf{x}_0, \mathbf{x}_1)$.

$T((g, \mathbf{M}, \mathbf{s}), y)$

1. Compute $\mathbf{M}^{-1}$ using $\mathbf{M}$.
2. Compute $g^{\mathbf{M}^{-1}\mathbf{M}\mathbf{x}_0} = g^{\mathbf{x}_0}$.
3. Take the discrete logarithm of each element of $g^{\mathbf{x}_0}$, yielding $\mathbf{x}_0$. Crucially, this is possible because the elements of $\mathbf{x}$ are in $\mathbb{Z}_d$ and $d = \mathsf{poly}\left(n\right)$, so the discrete logarithm can be computed in polynomial time by brute force.
4. Compute $\mathbf{x}_1 = \mathbf{x}_0 - \mathbf{s}$
5. Return $(\mathbf{x}_0, \mathbf{x}_1)$

5.7.3 Table of circuit sizes

A comparison of the resource requirements for computing $x^2 \bmod N$, for various problem sizes and circuit designs, is presented in Table 5.2. These counts are generated in the "abstract circuit" model, in which error correction, qubit routing, and other practical considerations are not included. For schoolbook and Karatsuba circuits, circuits are decomposed into a Clifford+T gate set. For the "phase" circuits, we allow controlled arbitrary phase rotations, as we expect these circuits to be appropriate for hardware (physical) qubits where these gates are native. Accordingly, we do not provide T gate counts for those circuits.

5.7.4 Cryptographic proofs of TCF properties

Here we prove the cryptographic properties of the trapdoor claw-free functions (TCFs) presented in the Methods section of the main text. We base our definitions on the Noisy

Circuit	Qubits	Gates (CCR_ϕ/ Toffoli allowed)	Gates (Clifford + T)	T Gates	Depth	Qubit measmts.
$n = 128$ (takes seconds on a desktop [174])						
Qubit-optimized phase	128	1.1×10^6	—	—	1.1×10^6	128
Gate-optimized phase	264	4.3×10^5	—	—	6.3×10^4	0
Schoolbook	515	1.4×10^5	9.1×10^5	3.9×10^5	1.9×10^4	3.5×10^4
Karatsuba	942	1.3×10^5	7.7×10^5	3.3×10^5	2.0×10^3	3.4×10^4
$n = 400$ (takes hours on a desktop [174])						
Qubit-optimized phase	400	$3.3 \times 10^{7*}$	—	—	$3.3 \times 10^{7*}$	400
Gate-optimized phase	812	$4.2 \times 10^{6*}$	—	—	$6.2 \times 10^{5*}$	0
Schoolbook	1603	1.3×10^6	8.7×10^6	3.6×10^6	5.9×10^4	3.3×10^5
Karatsuba	3051	8.8×10^5	5.4×10^6	2.3×10^6	5.3×10^4	2.4×10^5
$n = 829$ (record for factoring [175])						
Qubit-optimized phase	829	$3.0 \times 10^{8*}$	—	—	$2.9 \times 10^{8*}$	829
Gate-optimized phase	1671	$1.8 \times 10^{7*}$	—	—	$2.6 \times 10^{6*}$	0
Schoolbook	3319	5.6×10^6	3.8×10^7	1.6×10^7	$1.2 \times 10^{5*}$	1.4×10^6
Karatsuba	5522	3.0×10^6	1.8×10^7	7.7×10^6	$1.1 \times 10^{5*}$	8.0×10^5
$n = 1024$ (exceeds factoring record)						
Qubit-optimized phase	1024	$5.6 \times 10^{8*}$	—	—	$5.5 \times 10^{8*}$	1024
Gate-optimized phase	2061	$2.7 \times 10^{7*}$	—	—	$4.0 \times 10^{6*}$	0
Schoolbook	4097	8.3×10^6	5.7×10^7	2.4×10^7	$1.5 \times 10^{5*}$	2.1×10^6
Karatsuba	6801	4.3×10^6	2.6×10^7	1.1×10^7	$1.4 \times 10^{5*}$	1.1×10^6
Other algs. at $n = 1024$						
Rev. schoolbook †	8192	—	6.4×10^8	2.2×10^8	1.1×10^8	0
Rev. Karatsuba †	12544	—	5.7×10^8	1.9×10^8	2.4×10^7	0
Shor's alg. ‡	3100	—	—	$1.9 \times 10^{9*}$	—	—

Table 5.2: **Circuit sizes for various values of $n = \log N$.** Values may vary for different N of the same length. "Qubit-optimized phase" and "gate-optimized phase" refer to the circuits given in Figure 3(a) and 3(b) of the main text, respectively. "Qubit measmts." refers to the number of times qubits are measured and then reused during execution of the circuit. See Chapter 7 for alternative circuit constructions to the ones presented here. *From analytic estimate rather than building explicit circuit. †Reversible circuits constructed using Q# implementation of Ref. [176], and scaled to include Montgomery reduction. ‡Estimate from [177].

Trapdoor Claw-free Function family (NTCF) definition given in Definition 3.1 of Ref. [39], with certain modifications such as removing the adaptive hardcore bit requirement and the "noisy" nature of the functions.

We emphasize that in the definitions below, we define security only against classical attackers. Both the $x^2 \bmod N$ and DDH constructions could be trivially defeated by a quantum adversary via Shor's algorithm; since the purpose of the protocol in this paper is to demonstrate quantum capability, this type of adversary is allowed.

We also note that the TCF definition allows the 2-to-1 property to be "imperfect"—that is, we allow the fraction of pre-images which have a colliding pair to be less than 1. In the protocol, the verifier may simply discard any runs in which the prover supplied an output y value that is not part of a claw, that is, does not have two corresponding inputs. This will not affect the prover's ability to pass the classical threshold (since these runs are counted neither for or against the prover); it will only possibly affect the number of iterations of the protocol required to exceed the classical bound with the desired statistical significance. In the definition below we require the fraction of "good" inputs be at least a constant (which we set to 0.9); in principle the fraction could be as low as $1/\text{poly}(\lambda)$ without interfering with the protocol's effectiveness.

5.7.4.1 TCF definition

We use the following definition of a Trapdoor Claw-free Function family:

Definition 1. *Let λ be a security parameter, I a set of function indices, and X_i and Y_i finite sets for each $i \in I$. A family of functions*

$$\mathcal{F} = \{f_i : X_i \to Y_i\}_{i \in I}$$

is called a trapdoor claw free (TCF) family if the following conditions hold:

1. ***Efficient Function Generation.*** *There exists an efficient probabilistic algorithm* Gen *which generates a key $i \in I$ and the associated trapdoor data t_i:*

$$(i, t_i) \leftarrow \mathsf{Gen}(1^\lambda)$$

2. ***Trapdoor Injective Pair.*** *For all indices $i \in I$, the following conditions hold:*

   a) *Injective pair: Consider the set R_i of all tuples (x_0, x_1) such that $f_i(x_0) = f_i(x_1)$. Let $X_i' \subseteq X_i$ be the set of values x which appear in the elements of R_i. For all $x \in X_i'$, x appears in exactly one element of R_i; furthermore, there exists a value λ_c such that for all $\lambda > \lambda_c$, $|X_i'|/|X_i| > 0.9$.*

   b) *Trapdoor: There exists an efficient deterministic algorithm T such that for all $y \in Y_i$ and (x_0, x_1) such that $f_i(x_0) = f_i(x_1) = y$, $T(t_i, y) = (x_0, x_1)$.*

3. ***Claw-free.*** *For any non-uniform probabilistic polynomial time (nu-PPT) classical Turing machine $\mathcal{A}$, there exists a negligible function $\epsilon(\cdot)$ such that*

$$\Pr\left[f_i(x_0) = f_i(x_1) \wedge x_0 \neq x_1 | (x_0, x_1) \leftarrow \mathcal{A}(i)\right] < \epsilon(\lambda)$$

   where the probability is over both choice of i and the random coins of $\mathcal{A}$.

4. ***Efficient Superposition.*** *There exists an efficient quantum circuit that on input a key i prepares the state*

$$\frac{1}{\sqrt{|X_i|}} \sum_{x \in X_i} |x\rangle |f_i(x)\rangle$$

5.7.4.2 Proof of $x^2 \bmod N$ TCF

In this section we prove that the function family $\mathcal{F}_{\text{Rabin}}$ (defined in Methods) is a TCF by demonstrating each of the properties of Definition 1. Most of the properties follow directly from properties of the Rabin cryptosystem [132]; we reproduce several of the arguments here for completeness.

Theorem 4. *The function family $\mathcal{F}_{\text{Rabin}}$ is trapdoor claw-free, under the assumption of hardness of integer factorization.*

Proof. We demonstrate each of the properties of Definition 1:

1. **Efficient Function Generation.** Sampling large primes to generate p, q and N is efficient [132].

2. **Trapdoor Injective Pair.**

   a) Injective pair: By definition of the function, Y_i is the set of quadratic residues modulo N. For any $y \in Y_i$, consider the two values $a < p/2$ and $b < q/2$ such that $a^2 \equiv y \bmod p$ and $b^2 \equiv y \bmod q$. These values exist because y is a quadratic residue modulo pq, therefore it is also a quadratic residue modulo p and q. Define $c \equiv 1 \bmod p \equiv 0 \bmod q$ and $d \equiv 0 \bmod p \equiv 1 \bmod q$. The following four values x in the range $[0, N)$ have $x^2 \equiv y \bmod N$: $ac + bd, ac - bd, -ac + bd, -ac - bd$. Exactly two of these values are in the domain $[N/2]$ of the TCF, and constitute the injective pair; moreover, these two values will be unique as long as $a, b \neq 0$. Thus we may define the set $X_i' = \{x \in [N/2] | x \not\equiv 0 \bmod p \wedge x \not\equiv 0 \bmod q\}$. There exist exactly $((p-1)+(q-1))/2$ multiples of p or q in the set of integers $X_i = [N/2]$, thus $|X_i'|/|X_i| = 1 - ((p-1)+(q-1))/N$. Recall that p, q are defined to have length $\lambda/2$; if we let $\lambda_c = 12$, then $p, q > 2^5 = 32$. Since $1 - (31+31)/32^2 > 0.9$ and $|X_i'|/|X_i|$ increases monotonically with λ, we have $|X_i'|/|X_i| > 0.9$ for all $\lambda > \lambda_c$.

   b) Trapdoor: Because p and q were selected to have $p \equiv q \equiv 3 \bmod 4$, a and b in the expressions above can always be computed as $a = y^{(p+1)/4} \bmod p$ and $b = y^{(q+1)/4} \bmod q$, and then the preimages can be computed as defined above.

3. **Claw-free.** We show that knowledge of a claw x_0, x_1 can be used directly to factor N. Writing the claw as $(ac+bd, ac-bd)$ using the values a, b, c, d from above, we have $x_0 + x_1 = 2ac$. Because $c = 0 \bmod q$, $\gcd(x_0 + x_1, N) = q$ can be efficiently computed, which then also yields $p = N/q$. Thus, an algorithm that could be used efficiently to find claws could be equally used to efficiently factor N, which we assume to be hard.

4. **Efficient Superposition.** The set of preimages X_i is the set of integers $[N/2]$. A uniform superposition $\sum_{x \in X_i} |x\rangle$ may be computed by generating a uniform superposition of all bitstrings of length n (via Hadamard gate on every qubit), and then evaluating a

comparator circuit that generates the state $\sum |x\rangle |x < N/2\rangle$ where $|x < N/2\rangle$ is a bit on an ancilla. If this ancilla is then measured and the result is $|1\rangle$, the state is collapsed onto the superposition $\sum_{x \in X_i} |x\rangle$ (if the result is $|0\rangle$ the process should simply be repeated). Then a multiplication circuit to an empty register may be executed to generate the desired state $\sum_{x \in X} |x\rangle |x^2 \bmod N\rangle$.

□

5.7.4.3 Proof of Decisional Diffie-Hellman TCF

We now prove that $\mathcal{F}_{\text{DDH}}$ (defined in Methods) forms a trapdoor claw-free function family.

Theorem 5. *The function family $\mathcal{F}_{\text{DDH}}$ is trapdoor claw-free, under the assumption of hardness of the decisional Diffie-Hellman problem for the group $\mathbb{G}$.*

Proof. We demonstrate each of the properties of Definition 1:

1. **Efficient Function Generation.** Each step of Gen is efficient by inspection.

2. **Trapdoor Injective Pair.**

   a) Injective pair: First we note that the matrix $\mathbf{M}$ is chosen to be invertible, thus f_0 and f_1 are one-to-one. Therefore for all $\mathbf{x}_0 \in X_i$, at most one other preimage $\mathbf{x}_1 \in X_i$ has $f_i(\mathbf{x}_0) = f_i(\mathbf{x}_1)$. Furthermore, since colliding pairs have the structure $(0||\mathbf{x}'_0), (1||\mathbf{x}'_1)$ with $\mathbf{x}'_0 = \mathbf{x}'_1 + \mathbf{s}$ and $\mathbf{s} \in \{0,1\}^k$, the only preimages that will *not* form part of a colliding pair are those where $\mathbf{x}'_0$ has a zero element at an index where $\mathbf{s}$ is nonzero, or $\mathbf{x}'_1$ has an element equal to $d-1$ where $\mathbf{s}$ is nonzero (the vector element will be outside of the range of vector elements for the other vector). Thus $|X'_k|/|X_k| > (1 - 1/d)^k$. Since $d \sim \mathcal{O}(k^2)$ and $k \sim \mathcal{O}(\lambda)$, we have $\lim_{\lambda \to \infty} |X'_k|/|X_k| = 1$ with $|X'_k|/|X_k|$ monotonically increasing. Therefore, there exists a value λ_c such that $|X'_k|/|X_k| > 0.9$ for all $\lambda > \lambda_c$. (We note that if we set $k = \lambda$ and $d = k^2$, then $\lambda_c = 10$.)

   b) Trapdoor: The steps of the algorithm T are efficient by inspection. Crucially, the discrete logarithm of each vector element is possible by brute force, because the elements of $\mathbf{x}_0$ only take values up to polynomial in λ.

3. **Claw-free.** An algorithm which could efficiently compute a claw $(0||\mathbf{x}'_0, 1||\mathbf{x}'_1)$ could then trivially compute the secret vector $\mathbf{s} = \mathbf{x}'_0 - \mathbf{x}'_1$. For any matrix $\mathbf{M}'$, the existence of an algorithm to uniquely determine $\mathbf{s}$ from $(g^{\mathbf{M}'}, g^{\mathbf{M}'\mathbf{s}})$ would directly imply an algorithm for determining whether $\mathbf{M}'$ has full rank. But DDH implies it is computationally hard to determine whether a matrix $\mathbf{M}'$ is invertible given $g^{\mathbf{M}'}$ [130], [131]. Therefore DDH implies the claw-free property.

4. **Efficient Superposition.** Because d is a power of two, a superposition of all possible preimages $\mathbf{x}$ can be computed by applying Hadamard gates to every qubit in a register all initialized to $|0\rangle$. The function f can then be computed by a quantum circuit implementing a classical algorithm for the group operation of $\mathbb{G}$.

□

5.7.5 Overview of Trapdoor Claw-free Functions

In this section, we provide a brief overview of the cryptographic concepts upon which this work relies.

Foundational to the field of cryptography is the idea of a *one-way function*. Informally, this type of function is easy to compute, but hard to invert. Here, "easy" means that the function can be evaluated in time polynomial in the length of the input. By "hard" we mean that the cost of the best algorithm to invert the function is superpolynomial in the length of the input. In practice, for a given one-way function we desire that there exists a particular problem size (input length) for which the function can be evaluated fast enough that it is not overly costly to use, but for which inversion would be infeasible for even an adversary with large (but realistic) computing power. One way functions can be used directly to construct many useful cryptographic schemes, including pseudorandom number generators, private-key encryption, and secure digital signatures.

In this work, we rely on a specific type of one-way function called a *trapdoor claw-free function* (TCF). This class of functions has two additional features.

First, it has a *trapdoor*. This means that while the function is hard to invert in general, with the knowedge of some secret data (the trapdoor key) inversion becomes easy. This secret data should be easy to generate when the function is chosen (from a large family of similar functions), but should be hard to find given just the description of the function itself. For example, in this work we describe the function $x^2 \bmod N$, with N the product of two primes. The trapdoor is the factorization of N. It is easy to generate this function along with the trapdoor, by simply selecting two primes and multiplying them together. However, under the assumption of hardness of integer factorization, given only the function description (namely the value N) it is computationally hard to find the trapdoor (the factors p and q).

The second additional feature of a TCF is that it is *claw-free*. This means that the function is two-to-one (has two inputs that map to each output), but it is computationally hard to *find* two such colliding inputs without the trapdoor. Note that if it were possible to invert the function it would be trivial to find a collision (by picking an input, computing the function to get the output corresponding to it, and then inverting the function to find the second input mapping to that output). However the claw-free property is a bit stronger than the hardness of inversion: there exist some two-to-one functions which are one-way but not claw-free.

Importantly, in this work we only require that breaking the claw-free property is hard classically—indeed, the claw-free property of the DDH and $x^2 \bmod N$ TCFs described here

can be fully broken by quantum computers. However, perhaps surprisingly, we do not *require* that breaking the claw-free property is easy for a quantum machine. In fact, the claw-free property of the LWE and Ring-LWE based TCFs remains secure even against quantum attacks. This corresponds to a very powerful property of the protocol in this paper, and other related protocols: that a quantum computer can pass the test without actually being able to find a claw. This subtle distinction stems from the fact that the quantum prover generates a *superposition* over two inputs that collide. No measurement of such a state can yield both superposed values classically in full, but the test is designed to not require both values—just the results of an appropriate measurement of the superposition. A classical cheater, on the other hand, still cannot pass the test because the idea of a superposition does not exist classically.

5.7.6 Explanation of circuit complexities

Here we describe each of the asymptotic circuit complexities listed in Table I of the main text. For these estimates we drop factors of $\log\log n$ or less. In all cases, we assume integer multiplication can be performed in time $\mathcal{O}(n\log n)$ using the Schonhage-Strassen algorithm.

We emphasize that the value of n necessary to achieve classical hardness in practice varies widely among these functions, and also that the asymptotic complexities here may not be applicable at practical values of n.

LWE [39], [126] The LWE cost is dominated by multiplying an $\mathcal{O}(n\log n)\times n$ matrix of integers by a length n vector. The integers are of length $\log n$, so each multiplication is expected to take approximately $\mathcal{O}(\log n)$ time. Thus, the evaluation of the entire function requires $\mathcal{O}(n^2\log^2 n)$ operations.

x^2 mod N [132] The function can be computed in time $\mathcal{O}(n\log n)$ using Schonhage-Strassen multiplication algorithm and Montgomery reduction for the modulus.

Ring-LWE [41], [178]–[180] Ring-LWE is dominated by the cost of multiplying one polynomial by $\log n$ other polynomials. Through Number Theoretic Transform techniques similar to the Schonhage-Strassen algorithm, each polynomial multiplication can be performed in time $\mathcal{O}(n\log n)$, so the total runtime is $\mathcal{O}(n\log^2 n)$. We note that integer multiplication and polynomial multiplication can be mapped onto each other, so the runtimes for x^2 mod N and Ring-LWE scale identically except for the fact that Ring-LWE requires $\log n$ multiplications instead of $\mathcal{O}(1)$.

Diffie-Hellman [129]–[131] The Diffie-Hellman based construction defined in Methods requires performing multiplication of a $k\times k$ matrix by a vector, with $k\sim\mathcal{O}(n)$. However, the "addition" operation for the matrix-vector multiply is the group operation of $\mathbb{G}$; we expect this operation to have complexity at least $\mathcal{O}(n\log n)$ (for e.g. integer multiplication). The exponentiation operations have exponent at most $d\sim\mathcal{O}(k^2)$, so can be performed in $\mathcal{O}(\log n)$ group operations. So, for each of the k^2 matrix elements one must perform an operation of complexity $\mathcal{O}(n\log^2 n)$, yielding a total complexity of $\mathcal{O}(n^3\log^2 n)$.

Shor's Algorithm [119] Allowing for the use of Schonhage-Strassen integer multiplication, Shor's algorithm requires $\mathcal{O}(n^2\log n\log\log n)$ gates [181].

5.7.7 Optimal classical algorithm

Here we provide an example of a classical algorithm that saturates the probability bound of Theorem 2 of the main text. It has $p_x = 1$ and $p_{\text{CHSH}} = 3/4$.

For a TCF $f : X \to Y$, consider a classical prover that simply picks some value $x_0 \in X$, and then computes y as $f(x_0)$, without ever having knowledge of x_1. If the verifier requests a projective x measurement, they always return x_0, causing the verifier to accept with $p_x = 1$. In the other case (performing rounds 2 and 3 of the protocol), upon receiving r they compute $b_0 = x_0 \cdot r$. The cheating prover now simply assumes that $x_0 \cdot r = x_1 \cdot r$, and thus that the correct single-qubit state that would be held by a quantum prover is $|b_0\rangle$, and returns measurement outcomes accordingly. With probability $1/2$, $|b_0\rangle$ is in fact the correct single-qubit state; in this case they can always cause the verifier to accept. On the other hand, if $x_0 \cdot r \neq x_1 \cdot r$, the correct state is either $|+\rangle$ or $|-\rangle$. With probability $\frac{1}{2}$, the measurement outcome reported by the cheating prover will happen to be correct for this state too. Overall, this cheating prover will have $p_{\text{CHSH}} = (1 + \frac{1}{2})/2 = \frac{3}{4}$.

Thus we see $p_x + 4p_{\text{CHSH}} - 4 = 1 + 4 \cdot \frac{3}{4} - 4 = 0$ which saturates the bound.

5.7.8 Quantum circuits for Karatsuba and schoolbook multiplication

Classically, multiplication of large integers is generally performed using recursive algorithms such as Schonhage-Strassen [182] and Karatsuba which have complexity as low as $\mathcal{O}(n \log n)$. In the quantum setting, the need to store garbage bits at each level of recursion has limited their usefulness [183], [184]. There does exist a reversible construction of Karatsuba multiplication that uses a linear number of qubits [176], but due to overhead required for its implementation it does not begin to outperform schoolbook multiplication until the problem size reaches tens of thousands of bits.

Leveraging the irreversibility described in Section IID of the main text, we are able use these recursive algorithms directly, without needing to maintain garbage bits for later uncomputation. We implement both the $\mathcal{O}(n^{1.58})$ Karatsuba multiplication algorithm and the simple $\mathcal{O}(n^2)$ "schoolbook" algorithm. Due to efficiencies gained from discarding garbage bits, we find that the Karatsuba algorithm already begins to outcompete schoolbook multiplication at problem sizes of under 100 bits. Thus Karatsuba seems to be the best candidate for "full-scale" tests of quantum advantage at problem sizes of $n \sim 500 - 1000$ bits. We also note that the Schonhage-Strassen algorithm scales even better than Karatsuba as $\mathcal{O}(n \log n \log \log n)$. However, even in classical applications it has too much overhead to be useful at these problem sizes. We leave its potential quantum implementation to a future work.

The multiplication algorithms just described do not include the modulo N operation, it must be performed in a separate step. We implement the modulo using only two classical-quantum multiplications and one addition via Montgomery reduction [185]. Montgomery

reduction does introduce a constant R' into the product, but this factor can be removed in classical post-processing after $y = x^2 R' \bmod N$ is measured.

Finally, we note that at the implementation level, optimizing *classical* circuits for modular integer multiplication has received significant study in the context of performing cryptography on embedded devices and FPGAs [186]–[188]. Mapping such optimized circuits into the quantum context may be a promising avenue for further research.

5.7.9 Details of post-selection scheme

In this section we describe several details of the post-selection scheme proposed in Section IIC of the main text.

5.7.9.1 Quantum prover with no phase coherence saturates the classical bound

Consider the two states $|\psi_\pm\rangle = (|x_0\rangle \pm |x_1\rangle)_{\mathsf{x}} |y\rangle_{\mathsf{y}}$ for some claw (x_0, x_1) with $y = f_k(x_0) = f_k(x_1)$. Note that $|\psi_+\rangle$ is the state that would be held by a noise-free prover. Suppose a noisy quantum prover is capable of generating the mixed state

$$\rho_\delta = (1/2 + \delta) |\psi_+\rangle \langle\psi_+| + (1/2 - \delta) |\psi_-\rangle \langle\psi_-| . \tag{5.17}$$

In words, they are able to generate a state that is a superposition of the correct bitstrings, but with the correct phase only $1/2 + \delta$ fraction of the time. Here we show that such a prover can exceed the classical threshold of Theorem 2 of the main text, whenever $\delta > 0$. We proceed by examining this prover's behavior during the protocol.

First, we note that if the verifier requests a projective x measurement after Round 1 of the protocol, this prover will always succeed—they simply measure the x register as instructed, and the phase is not relevant. Thus, using the notation of Theorem 2, $p_x = 1$. With this value set, to exceed the bound we must achieve $p_{\text{CHSH}} > 3/4$. Naively performing the rest of the protocol as described in the main text does not exceed the bound when δ is small. However, the noisy prover can exceed the bound if they adjust the angle of their measurements in the third round of protocol (but preserve the sign of the measurement requested by the prover). We now demonstrate how.

Define $|\phi\rangle$ as the "correct" single-qubit state at the end of Round 2—one of $\{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$. Let $f_\updownarrow$ be the probability that our noisy prover holds the correct state when $|\phi\rangle \in \{|0\rangle, |1\rangle\}$, and $f_\leftrightarrow$ the corresponding probability when $|\phi\rangle \in \{|0\rangle, |1\rangle\}$. In the first case, the potential phase error of our prover does not affect the single-qubit state, so $f_\updownarrow = 1$. In the other case, the state is only correct when the phase is correct, so $f_\leftrightarrow = 1/2 + \delta$. We see that our prover will hold the correct single-qubit state with probability greater than $3/4$. But, if they naively measure in the prescribed off-diagonal basis $\theta \in \{\pi/4, -\pi/4\}$ from the verifier, for small δ their success probability will be less than $3/4$. This can be rectified by adjusting the rotation angle of the measurement basis.

Letting $\pm\theta'$ define the pair of measurement angles used by the prover in step 3 of the protocol (nominally $\theta' = |\theta| = \pi/4$), we can now express the prover's success probability

p_{CHSH} as

$$p_m = \frac{1}{2}\left[\cos^2\left(\frac{\theta'}{2}\right) f_{\updownarrow} + \cos^2\left(\frac{\theta'}{2} - \frac{\pi}{4}\right) f_{\leftrightarrow} + \sin^2\left(\frac{\theta'}{2}\right)(1 - f_{\updownarrow}) + \sin^2\left(\frac{\theta'}{2} - \frac{\pi}{4}\right)(1 - f_{\leftrightarrow})\right] \tag{5.18}$$

If the prover measures with $\theta' = \pi/4$ as prescribed in the protocol, the success rate will be $p_{\text{CHSH}} \approx 0.68 + \mathcal{O}(\delta) < 3/4$. However, if they instead adjust their measurement angle to $\theta = \delta$, they instead achieve $p_{\text{CHSH}} = 3/4 + 3\delta^2/8 - \mathcal{O}(\delta^3)$, which exceeds the classical bound (provided that δ is large enough to be noticeable).

In practice, both $f_{\updownarrow}$ and $f_{\leftrightarrow}$ are likely to be less than one; the optimal measurement angle can be determined as

$$\theta'_{\text{opt}} = \tan^{-1}\left(\frac{2f_{\leftrightarrow} - 1}{2f_{\updownarrow} - 1}\right) \tag{5.19}$$

which is the result of optimizing Equation 5.18 over θ'. In a real experiment, it would be most effective to empirically determine $f_{\updownarrow}$ and $f_{\leftrightarrow}$ and then use Equation 5.19 to determine the optimal measurement angle.

5.7.9.2 Details of simulation and error model

We now describe the details of the numerical simulation that was used to generate Figure 2 of the main text. For several values of the overall circuit fidelity $\mathcal{F}$, we established a per-gate fidelity as $f = \mathcal{F}^{1/N_g}$ where N_g is the number of gates in the $x^2 \bmod N$ circuit. We then generated a new circuit to compute the function $(3^a x)^2 \bmod 3^{2a}N$ for various values of a (see next subsection for an explanation of the choice $k = 3^a$). For each gate in the new circuit, with probability $1 - f$ we added a Pauli "error" operator randomly chosen from $\{X, Y, Z\}$ to one of the qubits to which the gate was being applied.

For the simulation, we randomly chose two primes p and q that multiplied to yield an integer N of length 512 bits. We then randomly chose a large set of colliding preimage pairs, and simulated the circuit separately for each such preimage (which is classically efficient, since the circuits only consist of X, CNOT, and Toffoli gates). The relative phase between each pair of preimages (due to error gates) was tracked explicitly during the simulation. Finally, the expected success rate of the prover was determined by analyzing the correctness of the bitstrings and their relative phase at the end of the circuit.

The primes p and q used to generate Figure 2 of the main text are (in base 10):

```
p = 113287732919697174280284729511923238986362403955638184856698528941220766063369
q = 983599673823371106353779572413533621838127094613863348191665028485127406927 27
```

5.7.9.3 Choice of $k = 3^a$ to improve postselection for $x^2 \bmod N$

In the previous subsection, we map the TCF $f_N = x^2 \bmod N$ to the function $f'_N = (kx)^2 \bmod k^2 N$. To achieve this at the implementation level, we may use essentially the same circuit for

modular multiplication; the only new requirement is to efficiently generate a superposition of multiples kx in the `x` register. We generate this superposition by starting with a uniform superposition over values x and then multiplying by k.

Normally, quantum multiplication circuits (like those we use to evaluate $x^2 \bmod N$) perform an out-of-place multiplication, where the result is stored in a new register. In this case, however, it is preferable to do the multiplication "in-place," where the result is stored in the input register itself—this way the y value is computed directly from the input register and thus is more likely to reflect errors that may occur in the input.

In general, performing in-place multiplication is complicated, particularly on a quantum register, because the input is being modified as it is being consumed (not to mention concerns about reversibility). However, multiplication by small constants is much simpler to implement. By setting k to a power of three, we are able to implement the in-place multiplication by performing a sequence of in-place multiplications by 3, which can each be performed quite efficiently (see implementation in the attached `Cirq` code [16]).

5.7.9.4 Theory prediction of Figure 2 of the main text

For the dashed "theory prediction" lines of Figure 2 of the main text, we predicted the success probabilities under two assumptions (which the numerical experiments are intended to test). First, among noisy runs where at least one bit flip error occurs, the output bitstring is approximately uniformly distributed. Second, we assume that with at least one phase flip error, the probability that the phase is correct in the final state is 1/2.

Under these assumptions, we compute the predicted success rates p_x and p_{CHSH} as follows:

1. For a given overall fidelity $\mathcal{F}$ of the original $x^2 \bmod N$ circuit containing N_g gates, compute a per-gate fidelity $f = \mathcal{F}^{1/N_g}$. Then compute the expected overall fidelity $\mathcal{F}'$ of running the slightly larger $(kx)^2 \bmod k^2N$ containing N'_g gates as $f^{N'_g}$.

2. Using $\mathcal{F}'$ and the given error model (see "Details of simulation and error model" section above), compute three disjoint probabilities: that no errors occur, that only phase errors occur, or that at least one bit flip error (and possibly also phase errors) occurs.

3. Compute the probability that the output will pass postselection, which includes both cases with no bit flip errors and those that are corrupted but happen to pass postselection by chance.

4. Normalizing to only those runs that pass postselection, compute p_x and p_{CHSH}:

    a) p_x is computed as the probability that no bit flip errors occurred (among those runs that pass postselection). This is a lower bound (that seems intuitively tight); it assumes a negligible probability that the measured pair (x, y) still has $y = f(x)$ despite bit flip errors.

[16] Code is available at `https://github.com/GregDMeyer/quantum-advantage` and is archived on Zenodo [155]

b) p_{CHSH} is computed by finding the probability that no errors occurred that would affect the single-qubit state at the end of round 2. When the correct single-qubit state should be polarized along Z, this is taken to be the probability that no bit flip errors occurred (phase errors are allowed since they will not affect this state); when the correct state should be polarized along X, it is taken as the probability that no errors at all have occurred. In these "no-error" cases, we compute the verifier's probability of accepting by applying the adjusted measurement basis described in the first sub-section above, "Quantum prover with no phase coherence saturates the classical bound". Finally, for the case that there was an error that could affect the single-qubit state, the probability that the verifier receives a correct measurement outcome is taken to be 1/2 (the single-qubit state is taken to be maximally mixed).

5. Compute the measure of "quantumness" from p_x and p_{CHSH}.

6. Compute the estimate runtime by multiplying the increase in quantum circuit size by the expected number of iterations required to pass postselection (which is computed from the analysis above).

Chapter 6

Implementing interactive protocols on a trapped-ion quantum computer

6.1 Introduction

To date, the field of experimental quantum computation has largely operated in a non-interactive paradigm, where classical data is extracted from the computation only at the very last step. While this has led to many exciting advances, it has also become clear that in practice, interactivity—made possible by mid-circuit measurements performed on the quantum device—will be crucial to the operation of useful quantum computers. For example, within quantum error correction, projective mid-circuit measurements are used to convert a continuum of possible errors into a specific discrete set of errors which can be corrected, as has been demonstrated in a recent experiment [151], [168]. Certain quantum machine learning algorithms also leverage mid-circuit measurements to introduce essential non-linearities [189]. Recent work has shown that interaction can do much more: it has emerged as an indispensable tool for verifying the behavior of untrusted quantum devices [39], [47], [48], and even for testing the fundamentals of quantum mechanics itself [120].

Consider the scenario of a classical computer sending commands to an untrusted quantum device that it cannot feasibly simulate. This could consist of a lab computer testing a new, large quantum device, but also perhaps a user connecting to a quantum cloud computing service over the internet. At first sight, the inability of the classical machine to simulate the quantum one seems to pose a difficulty for certifying the output. This challenge mirrors one explored in the field of classical computer science, which asks whether a skeptical, computationally-bounded "verifier," who is not powerful enough to validate a given statement on their own, can be convinced of its veracity by a more powerful but untrusted "prover." Several decades ago, this idea began to be pursued through a novel tool called an interactive proof. In these protocols, the verifier's goal is to accept only valid statements, regardless of whether the prover behaves honestly or attempts to cheat. One of the greatest achievements of computational complexity theory is a set of results showing

that in certain scenarios multiple rounds of interaction allow the verifier to detect cheating by even arbitrarily computationally powerful provers [190]–[192]. The essential idea is that interaction can force the prover to commit to some piece of information early in the protocol, upon which the verifier follows up with queries that can only be answered consistently if the prover is being truthful. In exciting recent developments, success has been found in the application of this idea to quantum computing: interactive proofs have been shown to allow the verification of a number of practical quantum tasks, including random number generation, [39] remote quantum state preparation, [48] and the delegation of computations to an untrusted quantum server.[47] Perhaps the most direct application of an interactive protocol is for a "cryptographic proof of quantumness"—a protocol that allows a quantum device to convincingly demonstrate its non-classical behavior to a polynomial-time classical verifier, by performing a task that is assumed to be computationally hard for a classical machine yet is efficient to check [39], [41], [115].

The simplest proof of quantumness in general is a Bell test (which does not rely on a computational hardness assumption) [127]. It uses entanglement to generate correlations that would be impossible to classically reproduce without communication. While the Bell test's simplicity is attractive, avoiding the communication loophole requires the use of multiple quantum devices which are separated by considerable distance. [193]–[195] In order to prove the quantumness of a single "black-box" quantum device whose inner workings are hidden from the verifier, one can instead rely on differences in classical and quantum computational power—in other words, asking the device to demonstrate quantum computational advantage. In contrast to recent sampling-based tests of quantum computational advantage [15], [17], [18], [20], [21], [32]–[35], [114], in a cryptographic proof of quantumness the verification step must also be efficient. While in principle any algorithm that exhibits a quantum speedup and has an efficiently-verifiable output could be used for this purpose, most such experiments are infeasible today because the necessary circuits are far too large to run successfully on current quantum computers. Remarkably, it has been shown that interactive proofs provide a way to reduce the experimental cost (in qubits and gate depth) of this type of test, while maintaining efficient verification and classical hardness.

In practice, the experimental implementation of interactivity is extremely challenging. It requires the ability to independently measure subsets of qubits in the middle of a quantum circuit and to continue coherent evolution afterwards. Unfortunately, the measurement of a target qubit typically disturbs neighboring qubits, degrading the quality of computations following the mid-circuit measurement. One solution, which finds commonality among atomic quantum computing platforms, is to spatially isolate target qubits via shuttling [196]–[198]. While daunting from the perspective of quantum control, experimental progress toward coherent qubit shuttling opens the door not only to interactivity but also to distinct information processing architectures [199].

In this work, we implement two complementary interactive cryptographic proof of quantumness protocols, shown in the schematic of Fig. 6.1, on an ion trap quantum computer with up to 11 qubits using circuits with up to 145 gates. The interactions between verifier and prover are enabled by the experimental realization of mid-circuit measurements on a

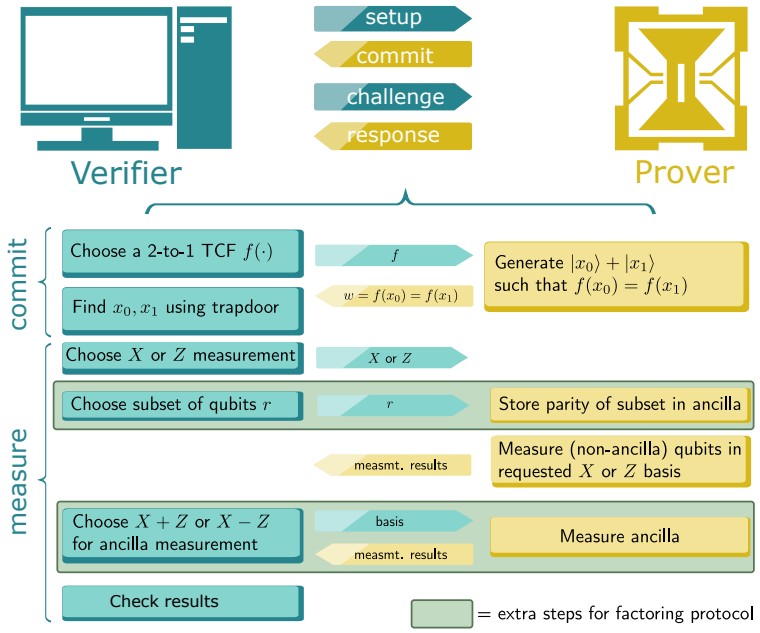


Figure 6.1: **Schematic of an interactive quantum verification protocol.** The verifier's goal is to test the "quantumness" of the prover through an exchange of classical information. The protocol begins with the verifier sending the prover an instance of a trapdoor claw-free function. By applying this function to a superposition of all possible inputs and projectively measuring the result, the prover commits to a particular quantum state $|x_0\rangle + |x_1\rangle$. Subsequent challenges issued by the verifier specify how to measure this state and enable the efficient validation of the prover's commitment. The LWE-based protocol requires two rounds of interaction, while the factoring-based protocol requires an additional round (green box).

portion of the qubits (Fig. 6.2) [151], [198], [200]. The first protocol involves two rounds of interaction and is based upon the learning with errors (LWE) problem [126], [201]. The LWE construction is unique because it exhibits a property known as the "adaptive hardcore bit" [39] (described in more detail in the next section), which enables a particularly simple measurement scheme. The second protocol is the one introduced in Chapter 5, which circumvents the need for this special property and thus applies to a more general class of cryptographic functions. By using an additional round of interaction, the cryptographic information is condensed onto the state of a single qubit. This makes it possible to implement a cryptographic proof of quantumness which is as hard to spoof classically as factoring, but whose associated circuits can exhibit an asymptotic scaling much simpler than Shor's algorithm ($\mathcal{O}(n \log n)$ instead of $\mathcal{O}(n^2 \log n)$, in terms of gate counts)[115].

6.2 Trapdoor claw-free functions

Both interactive protocols (Fig. 6.1) rely upon a cryptographic primitive called a trapdoor claw-free function (TCF) [202]—a 2-to-1 function f for which it is cryptographically hard to find two inputs mapping to the same output. Such pairs of colliding inputs are called "claws", and the term "claw-free" refers to the hardness of finding them. The function also has a "trapdoor," a secret key with which it is easy to compute the inputs x_0 and x_1 from any output $w = f(x_0) = f(x_1)$. The intuition behind the protocols is the following: Despite the claw-free property, a quantum computer can efficiently generate a superposition of two inputs that form a claw; this is most simply realized by evaluating f on a superposition of the entire domain, and then collapsing to a single output, w, via measurement. In this way, a quantum prover can generate the state $|\psi\rangle = (|x_0\rangle + |x_1\rangle)\,|w\rangle$, where w is the measurement result. The prover now sends w to the verifier, who then uses the trapdoor to compute x_0 and x_1, thus giving the verifier full knowledge of the prover's quantum state. The verifier then asks the prover to measure $|\psi\rangle$. In particular, they request either a standard basis measurement (yielding x_0 or x_1 in full), or a measurement that interferes the states $|x_0\rangle$ and $|x_1\rangle$. (Note that the value of w, and by association x_0 and x_1, changes each time the protocol is executed, so it is not possible to find a collision (x_0, x_1) by simply repeating this process with a standard basis measurement multiple times). The verifier checks the measurement result on a per-shot basis. Crucially, consistently producing correct values for these measurements results is impossible for a classical prover (assuming they cannot find a claw of the TCF), so reliably returning correct results constitutes a proof of quantumness.

6.2.1 The learning with errors problem

It is believed to be classically intractable to recover an input vector from the result of certain noisy matrix-vector multiplications—this constitutes the LWE problem [126], [201]. In particular, a secret vector, $s \in \{0,1\}^n$, can be encoded into an output vector, $y = As + e$, where $A \in \mathbb{Z}_q^{m\times n}$ is a matrix and e is an error vector corresponding to the noise. Using the LWE problem, a TCF can be constructed as $f(b, x) = \lfloor Ax + b \cdot y \rceil$, where b is a single bit that controls whether y gets added to Ax and $\lfloor\cdot\rceil$ denotes a rounding operation [203], [204] (see Methods section 6.6.7 for additional details). Here, s and e play the role of the trapdoor, and a claw corresponds to colliding inputs $\{(0, x_0), (1, x_1)\}$ with $f(0, x_0) = f(1, x_1)$ and $x_0 = x_1 + s$. By implementing the protocol described above and illustrated in Figure 6.1, the prover is able to generate the state $|\psi\rangle = (|0, x_0\rangle + |1, x_1\rangle)\,|w\rangle$. For the aforementioned "interference" measurement, the prover simply measures each qubit of the superposition in the X basis. Crucially, the result of this measurement is cryptographically protected by the adaptive hardcore bit property, which is a strengthening of the claw-free property [39]. Informally, it says that for any input x_0 (of the prover's choosing), it is cryptographically hard to determine even a single bit of information about x_1 (as opposed to the entire value, which is the guarantee of the claw-free assumption).

6.2.2 Rabin's function

The function $f(x) = x^2 \bmod N$, with N being the product of two primes, was originally introduced in the context of digital signatures [132], [133]. This function has the property that finding two colliding inputs (a claw) in the range $[0, N/2]$ is as hard as factoring N. Moreover, the prime decomposition $N = pq$ can serve as a trapdoor, enabling one to invert the function for any output. Thus, $f(x)$ is a trapdoor claw-free function. However, $f(x)$ does not have the adaptive hardcore bit property, making the simple X-basis "interference" measurement (described in the LWE context above) not provably secure. To get around this, we perform the "interference" measurement differently. First, the verifier chooses a random subset of the qubits of the superposition, and the prover stores the parity of that subset on an ancilla. Then, the prover measures everything except the ancilla in the X basis. Given our cryptographic assumption that the prover cannot find a claw, the prover cannot guess the polarization of the remaining ancillary qubit. This is directly analogous to how, in Bell experiments, the assumption of no-signaling-faster-than-light implies that if Alice measures one half of an EPR pair, a space-like separated Bob who holds the other half is unable to immediately guess its polarization. Following this intuition, the verifier requests a measurement of the ancilla qubit in the $Z + X$ or $Z - X$ basis, effectively completing the Bell test [127], [128]; the verifier accepts if the prover returns the more likely measurement outcome. Crucially, the dependence of the measurement result on the claw renders it infeasible to guess classically [115].

6.3 Implementing an interactive cryptographic proof

In order to implement an interactive cryptographic proof of quantumness, we design quantum circuits for both the LWE- and factoring-based protocols. The high-level circuit diagrams are shown in Figs. 6.3(a,b). In both cases, the circuits are composed of several sections. First, the prover creates a uniform superposition $|\psi\rangle = \sum_{x=0}^{2^n-1} |x\rangle$ via Hadamard gates, where n is the number of input qubits. Then, they compute the TCF on an output register using this superposition as input [Fig. 6.3(a,d)], thereby generating the state $|\psi\rangle = \sum_x |x\rangle |f(x)\rangle$. Next, the prover performs a mid-circuit measurement on the output register, collapsing the state to $|\psi\rangle = (|x_0\rangle + |x_1\rangle) |w\rangle$. Finally, based on the verifier's choice of measurement scheme (i.e. standard vs. interference), the prover must perform additional coherent gates and measurements (see Methods for a full description of the quantum circuits used).

We implement both interactive protocols using an ion trap quantum computer, with a base chain length of 15 ions (Fig. 6.2); for each $^{171}Yb^+$ ion, a qubit is encoded in a pair of hyperfine levels [205]. The quantum circuits are implemented via the consecutive application of native single and two-qubit gates using individual optical addressing [Fig. 6.2(a)] [206]. In order to realize rapid successive two-qubit interactions, we position the ions in a single, closely-spaced linear chain [Fig. 6.2(d)].

This geometry makes it challenging to implement mid-circuit measurements, because

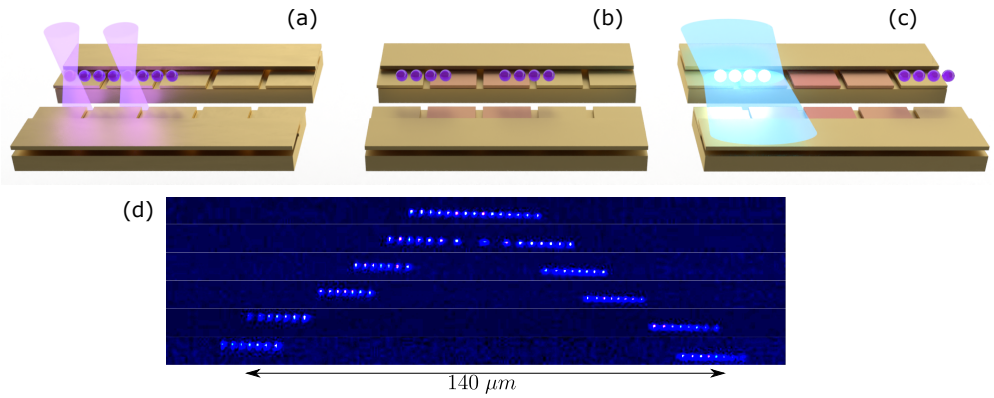


Figure 6.2: **Mid-circuit measurements with shuttling.**(a-c) Schematic illustration of our mid-circuit measurement protocol. (a) To start, the ions are closely spaced in a 1D chain above a surface trap. Coherent gates are implemented via a combination of individual addressing beams (purple) and global beams (not shown). Both the coherent addressing beams and the detection optics are aligned to ions at the same section of the trap. (b) By tuning the electrodes of the surface trap, we can adjust the potential to deterministically split the ion chain. Depending on the protocol, we split the chain into either two or three individual segments. We optimize the rate of shuttling to minimize the perturbation of the motional state. (c) Once the segments are sufficiently far away from one another, it is possible to measure (blue beam) an individual segment without disturbing the coherence of the remaining ions. After the measurement, the shuttling is reversed and the ion chain is recombined. (d) Fluorescence images of an example shuttling protocol for a chain of $N = 15$ ions. At the start, the average spacing between ions is $\sim 4\mu$m. At the end of the splitting procedure, the distance between the two segments is $\sim 550\mu$m um. The images show the splitting up to a distance of $\sim 140\mu$m, at which point the two sub-chains reach the edge of the detection beam.

light scattered from nearby ions during a state-dependent fluorescence measurement can destroy the state of the other ions. To overcome this issue, we vary the voltages on the trap electrodes to split and shuttle the ion chain, thereby spatially isolating the ions not being measured (Fig. 6.2a-c). Depending on the protocol, the ion chain is split into either two or three segments. To measure the ions in a particular segment, we re-shape the electric potential to align the target segment with the detection system. In addition, we calibrate and correct for spatial drifts of the optical beams, variations of stray fields, and unwanted phase accumulation during shuttling (see Methods sections 6.6.2, 6.6.5 for additional details).

In this demonstration, the qubits play the role of the prover and the classical control system plays the role of the verifier. This allows us to compile the decisions of the verifier into the classical controller prior to execution of the quantum circuit.

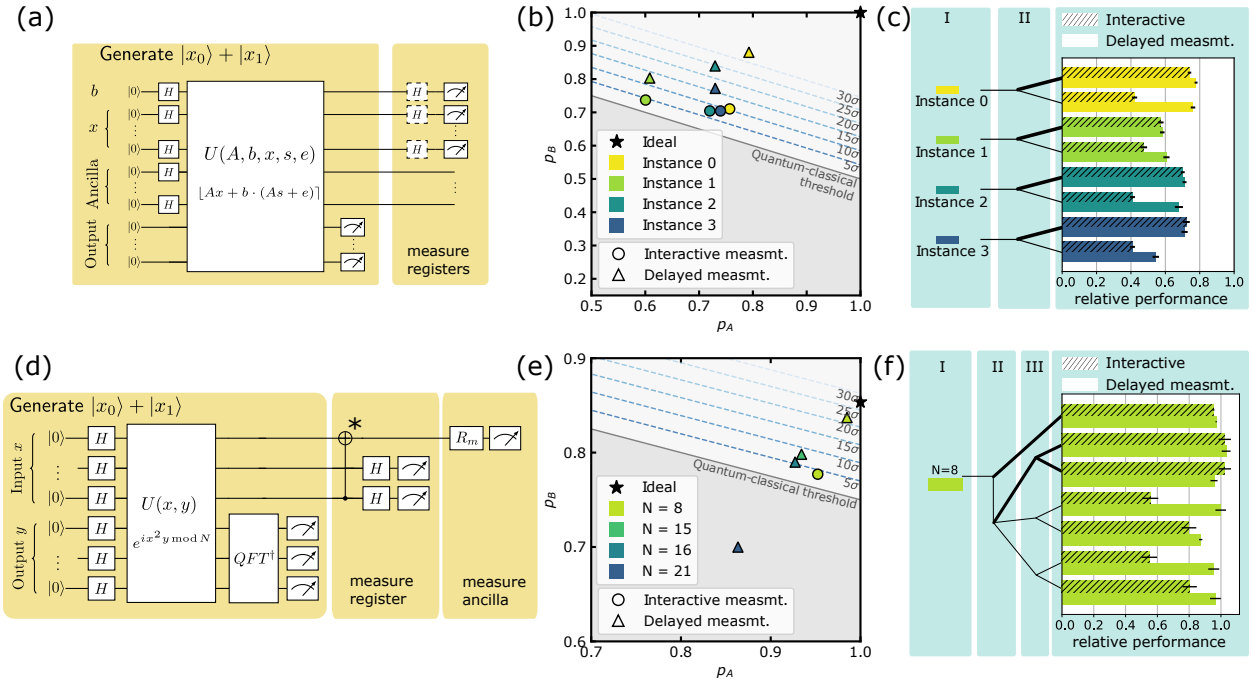


Figure 6.3: **Circuit and results of experiment impelmentations.**(a),(d) depict the circuit diagrams for the LWE- and factoring-based protocols, respectively. Details about the implementation of $U(A, b, x, y)$ and $U(x, y)$ are provided in the Methods section 6.6.7. In (d), the CNOT gate marked with an asterisk represents the operations needed to store the parity of selected qubits in the ancilla. To reduce the impact of shuttling-induced gate fidelity degradation, we compute the parity for all of the verifier's possible selections and then choose the relevant one once the prover receives the challenge. (b),(e): Experimentally measured probabilities of passing the standard-basis (p_A) and interference measurement (p_B) challenges for the LWE- and factoring-based protocols. These probabilities are compared against the asymptotic classical limits ($p_A + 2p_B \leq 2$ for LWE, derived in the Methods section 6.6.10, and $p_A + 4p_B \leq 4$ for factoring [115]). Results for both interactive and delayed-measurement version of the protocols are presented. Numerical values of p_A and p_B for each experiment, and the corresponding values of statistical significance, are provided in the Methods section 6.6.1. (c),(f): The relative performance, R, of the experiments for all possible branches. Certain branches (thick lines) are robust to phase errors and exhibit similar performance for both interactive and delayed-measurement protocols.

6.4 Beating the classical threshold

As in a Bell test, even a classical prover can pass the verifier's challenges with finite probability. If the classical prover cannot find a claw in the TCF (which is assumed to be the case for sufficiently large problem sizes), this probability can be bounded by an asymptotic "classical threshold"—which a quantum prover must exceed to demonstrate advantage.

(For a discussion of what it means that this threshold is "asymptotic" rather than absolute, see the Methods section 6.6.9). For both protocols, this threshold is best expressed in terms of the probabilities of passing the verifier's "standard basis" and "interference" checks, which we denote as p_A and p_B, respectively (see Methods section 6.6.3 for the definition of the verifier's checks). For the LWE-based protocol, the classical threshold is given by $p_A + 2p_B - 2 \leq \epsilon$ (derivation in Methods section 6.6.10); for the factoring-based protocol, it is given by $p_A + 4p_B - 4 \leq \epsilon$. [115] In both cases, ϵ is a function which goes to zero exponentially in the problem size. An intuition for the difference between the thresholds is that the factoring-based protocol requires an additional round of interaction during the "interference" test.

As depicted in Figure 6.3(b), we perform multiple instances of the LWE-based protocol for different matrices A and noise vectors e. For each of the verifier's possible choices, we repeat the experiment $\sim 10^3$ times to collect statistics. This yields the experimental probabilities p_A and p_B, allowing us to confirm that the quantum prover exceeds the asymptotic classical threshold in all cases. The statistical significance by which the bound is exceeded (more than 6σ in all cases, see Table 6.2 in the Methods section 6.6.1) is shown in Figure 3(b). Figure 6.3(e) depicts the analogous results for the factoring-based protocol, where the different instances correspond to different values of N. For all but $N = 21$, which requires the deepest circuit, the results exceed the asymptotic classical bound with more than 4σ statistical significance. We utilize an error-mitigation strategy based on excluding iterations where w is measured to be invalid, i.e. not in the range of f (see 6.6.4); effectively, this implements a post-selection which suppresses bit-flip errors [115].

To further analyze the performance of each branch of the interactive protocol, corresponding to the verifier's choices [Figs. 6.3(c,f)], we define the relative performance $R = (p_{\text{exp}} - p_{\text{guess}})/(p_{\text{ideal}} - p_{\text{guess}})$ for each branch, where p_{ideal} is the probability that an error-free quantum prover would pass, p_{guess} is the probability that a random guesser would pass, and p_{exp} is the passing rate measured in the experiment. This criterion is a way of isolating and evaluating the effect of noise on the success probabilities of each branch, by removing effects such as the fact that an error-free run may still happen to get rejected by the verifier, which is inherent to the protocol. In particular, for a perfect (noise-free) quantum prover $R = 1$ always, and for a device so noisy that its outputs are uniformly random $R = 0$. To probe the noise effects of the mid-circuit measurements, we implement two versions of the protocols: one interactively (the normal protocol) and another with all measurements delayed until the end of the circuit, and compare the relative performance between the two cases. We emphasize that the delayed-measurement version is only a tool to probe our experimental system, and may be vulnerable to classical spoofing even if it were run at large problem sizes where the other cryptographic assumptions hold—the interaction enabled by mid-circuit measurements is crucial.

For the LWE-based protocol, there are two rounds of interaction, corresponding to the two branches, I and II shown in Fig. 6.3(c), while for the factoring-based protocol there are three rounds of interaction [Fig. 6.3(f)]. By comparing the relative performance between the interactive and delayed-measurement versions of our experiment, we are able to probe a

subtle feature of the protocols—namely, that certain branches are robust to additional decoherence induced by the mid-circuit measurements. Microscopically, this robustness arises because these branches (thick lines, Figs. 6.3c,f) do not depend on the phase coherence between $|x_0\rangle$ and $|x_1\rangle$. In particular, this is true for the standard-basis measurement branches in both protocols, and also for the branches of the factoring-based protocol where the ancilla is polarized in the Z basis (see Methods section 6.6.6). Noting that mid-circuit measurements are expected to induce mainly phase errors, one would predict that those branches insensitive to phase errors should yield similar performance in both the interactive and delayed-measurement cases. This is indeed borne out by the data.

6.5 Discussion and outlook

There are two main experimental challenges to demonstrating quantum computational advantage via interactive protocols: 1) integrating mid-circuit measurements into arbitrary quantum circuits, with sufficiently high overall fidelity to pass the verifier's tests, and 2) scaling the protocols up to large enough problem sizes that it is classically infeasible to break the cryptographic assumptions. In this work we have overcome the first obstacle, successfully implementing two interactive cryptographic proofs of quantumness with high enough fidelity to pass the verifier's challenges. We leave the second challenge, of scaling these protocols up, to a future work. We estimate that one should be able to perform a cryptographic proof of quantum computational advantage using ~ 1600 qubits (see Methods section 6.6.13). Note that while this qubit count is comparable to some implementations of Shor's algorithm, the circuits are orders of magnitude smaller in gate count ($\mathcal{O}(n \log n)$ vs. $\mathcal{O}(n^2 \log n)$) and depth [115]. Even with those smaller circuits, the challenge on near-term devices will almost certainly remain the circuit depth; interestingly, recent advances suggest that our interactive protocols can be performed in constant depth at the cost of a larger number of qubits [147], [148]. Once this scaling is achieved in an experiment, it will demonstrate directly-verifiable quantum computational advantage. This would mark a new step forward from recent sampling experiments, which have demonstrated the system sizes and fidelities necessary to make classical simulation extremely hard or impossible [15], [17], [18], [20], [21], [32]–[35], [114] but have no method to directly and efficiently verify the output (and moreover, practical strategies for a classical impostor to replicate the sampling are still being explored [24]–[31]).

Our work also opens the door to a number of other intriguing directions. A clear next step is to apply the power of quantum interactive protocols to achieve more than just quantum advantage—for example, pursuing such tasks as certifiable random number generation, remote state preparation and the verification of arbitrary quantum computations [39], [47], [48]. We emphasize that unlike e.g. Bell-test based protocols for random number generation, interactive proofs allow us to perform these cryptographic tasks with a single "black-box" prover with which the verifier can only interact classically. This has the potential to allow these types of protocols (including our cryptographic proofs of quantumness) to be performed

on a remote prover, such as a quantum cloud service on the internet, enabling a wide variety of practical applications. Finally, the advent of mid-circuit measurement capabilities in a number of platforms [198], [200], [207], [208], enables the exploration of new phenomena such as entanglement phase transitions [209]–[211] as well as the demonstration of coherent feedback protocols including quantum error correction [151].

6.6 Additional proofs and data

6.6.1 Result data

In Tables 6.1 and 6.2 we present the numerical results for each configuration of the experiment, along with the number of samples obtained (N_A and N_B), the measure of quantumness q, and the statistical significance of the result (see Methods Section 6.6.11 for a description of how the significance is calculated).

We note that for the computational Bell test protocol, the sample size N_B is less than the actual number of shots that passed postselection (ultimately leading to slightly less statistical significance than might otherwise be expected). This is because the sample size varied for different values of the verifier's string r, yet we are interested in the passing rate p_B averaged uniformly over all r (not weighted by number of shots). To account for this, we simply took the r-value with the fewest number of shots, and computed N_B as if every r value had had that sample size (even if some values of r had more).

We also note that in some cases the statistical significance denoted here may be higher than that visually displayed in Figure 6.3 of the main text; this is because the contour lines in that figure correspond to the configuration with the smallest sample size.

N	Measurement scheme	p_A	p_B	N_A	N_B	Quantumness q	Significance
8	interactive	0.952	0.777	4096	15267	0.061	4.3σ
8	delayed	0.985	0.837	2736	17361	0.334	24.1σ
15	delayed	0.934	0.798	2361	31353	0.127	10.0σ
16	delayed	0.927	0.790	3874	53550	0.087	8.8σ
21	delayed	0.864	0.700	2066	27944	-0.338	—

Table 6.1: **Results for various configurations of the computational Bell test protocol.** For this protocol $q = p_A + 4p_B - 4$.

6.6.2 Trapped ion quantum computer

The trapped ion quantum computer used for this study was designed, built, and operated at the University of Maryland and is described elsewhere [206], [212]. The system consists

Instance	Measurement scheme	p_A	p_B	N_A	N_B	Quantumness q	Significance
0	interactive	0.757	0.710	8000	13381	0.178	18.6σ
0	delayed	0.793	0.880	10000	9415	0.553	60.3σ
1	interactive	0.601	0.737	8000	7622	0.075	6.2σ
1	delayed	0.608	0.803	8000	7547	0.215	18.0σ
2	interactive	0.720	0.704	14000	15310	0.129	15.0σ
2	delayed	0.730	0.839	4000	3735	0.409	24.6σ
3	interactive	0.740	0.704	8000	15189	0.148	16.2σ
3	delayed	0.730	0.772	8000	7528	0.274	23.1σ

Table 6.2: **Results for various configurations of the LWE-based protocol.** For this protocol $q = p_A + 2p_B - 2$.

of a chain of fifteen single $^{171}Yb^+$ ions confined in a Paul trap and laser cooled close to their motional ground state. Each ion provides one physical qubit in the form of a pair of states in the hyperfine-split $^2S_{1/2}$ ground level with an energy difference of 12.642821 GHz, which is insensitive to magnetic fields to first order. The qubits are collectively initialized through optical pumping, and state readout is accomplished by state-dependent fluorescence detection [213]. Qubit operations are realized via pairs of Raman beams, derived from a single 355-nm mode-locked laser [214]. These optical controllers consist of an array of individual addressing beams and a counter-propagating global beam that illuminates the entire chain. Single qubit gates are realized by driving resonant Rabi rotations of defined phase, amplitude, and duration. Single-qubit rotations about the z-axis, are performed classically with negligible error. Two-qubit gates are achieved by illuminating two selected ions with beat-note frequencies near motional sidebands and creating an effective Ising spin-spin interaction via transient entanglement between the two ion qubits and all modes of motion [215]–[217]. To ensure that the motion is disentangled from the qubit states at the end of the interaction, we used a pulse shaping scheme by modulating the amplitude of the global beam [218].

6.6.3 Verifier's checks

In this section we explicitly state the checks performed by the verifier to decide whether to accept or reject the prover's responses for each run of the protocol. We emphasize that these checks are performed on a per-shot basis, and the empirical success rates p_A and p_B are defined as the fraction of runs (after postselection, see below) for which the verifier accepted the prover's responses.

For both protocols, the "A" or "standard basis" branch check is simple. The prover has

already supplied the verifier with the output value w; for this test the prover is expected to measure a value x such that $f(x) = w$. Thus in this case the verifier simply evaluates $f(x)$ for the prover's supplied input x and confirms that it is equal to w.

For the "B" or "interference" measurement, the measurement scheme and verification check is different for the two protocols. For the LWE-based protocol, the interference measurement is an X-basis measurement of all of the qubits holding the input superposition $|x_0\rangle + |x_1\rangle$. This measurement will return a bitstring d of the same length as the number of qubits in that superposition, where for each qubit, the corresponding bit of d is 0 if the measurement returned the $|+\rangle$ eigenstate and 1 if the measurement returned the $|-\rangle$ eigenstate. The verifier has previously received the value w from the prover and used the trapdoor to compute x_0 and x_1; the verifier accepts the string d if it satisfies the equation

$$d \cdot x_0 = d \cdot x_1 \tag{6.1}$$

where $(\cdot)$ denotes the binary inner product, i.e. $a \cdot b = \sum_i a_i b_i \mod 2$. It can be shown that a perfect (noise-free) measurement of the superposition $|x_0\rangle + |x_1\rangle$ will yield a string d satisfying Eq. 6.1 with probability 1.

The interference measurement for the computational Bell test involves a sequence of two measurements (in addition to the first measurement of the string w). The first measurement yields a bitstring d as above. After performing that measurement, the prover holds the single-qubit state $(-1)^{d \cdot x_0} |r \cdot x_0\rangle + (-1)^{d \cdot x_0} |r \cdot x_0\rangle$, where $(\cdot)$ is the binary inner product as above and r is a random bitstring supplied by the verifier. This state is one of $\{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$, and is fully known to the verifier after receiving d (via use of the trapdoor to compute x_0 and x_1). The second measurement is of this single qubit, in an intermediate basis $Z+X$ or $Z-X$ chosen by the verifier. For any of the four possible states, one eigenstate of the measurement basis will be measured with probability $\cos^2(\pi/8) \approx 85\%$ (with the other having probability $\sim 15\%$), just as in a Bell test. The verifier accepts the measurement result if it corresponds to this more-likely result; an ideal (noise-free) prover will be accepted with probability $\sim 85\%$ (see Figure 6.3 of the main text).

6.6.4 Post-selection

Both the factoring-based and LWE-based protocols involve post-selection on the measurement results throughout the experiment.

For the factoring-based protocol, this post-selection is performed on the measured value of the output register w. Due to quantum errors in the experiment, in practice it is possible to measure a value of w that does not correspond to any inputs of the TCF—that is, there do not exist x_0, x_1 for which $f(x_0) = f(x_1) = w$, due to noise. Because such a result would not be possible without errors, measuring such a value indicates that a quantum error has occurred [115]. Thus, we perform post-selection by discarding all runs for which the measured value w does not have two corresponding inputs.

On the other hand, for the LWE protocol, we post-select in order to satisfy the conditions for the adaptive hardcore bit property to hold, as without this property, the protocol could

Instance	Delayed Measurement	Interactive Measurement
0	3753/4000	13381/14000
1	7547/8000	7622/8000
2	3735/4000	15310/16000
3	7528/8000	15144/16000

Table 6.3: **Fractions of runs kept during post-selection for the LWE-based protocol, in the "interference" measurement branch.** All runs are kept for the standard basis measurement.

N	Interactive	Branch	Runs kept/Total
8	Yes	A	4096/9000
8	Yes	B, r=01	5093/12000
8	Yes	B, r=10	5089/12000
8	Yes	B, r=11	5492/12000
8	No	A	2736/6000
8	No	B, r=01	5787/12000
8	No	B, r=10	5818/12000
8	No	B, r=11	5865/12000
15	No	A	2361/6000
15	No	B, r=001	4636/12000
15	No	B, r=010	4532/12000
15	No	B, r=011	4666/12000
15	No	B, r=100	4496/12000
15	No	B, r=101	4727/12000
15	No	B, r=110	4479/12000
15	No	B, r=111	4673/12000

N	Interactive	Branch	Runs kept/Total
16	No	A	3874/6000
16	No	B, r=001	7842/12000
16	No	B, r=010	7847/12000
16	No	B, r=011	7732/12000
16	No	B, r=100	7936/12000
16	No	B, r=101	7870/12000
16	No	B, r=110	7841/12000
16	No	B, r=111	7650/12000
21	No	A	2066/6000
21	No	B, r=001	3992/12000
21	No	B, r=010	4273/12000
21	No	B, r=011	4137/12000
21	No	B, r=100	4182/12000
21	No	B, r=101	4193/12000
21	No	B, r=110	4261/12000
21	No	B, r=111	4221/12000

Table 6.4: **Fraction of runs kept during postselection for each branch of the factoring-based protocol.**

be susceptible to attacks. In particular, the adaptive hardcore bit property requires that the result obtained from measuring the x register using the "interference" measurement scheme be a nonzero bitstring [39]. Hence, we simply post-select on this condition for the LWE case. Tables 6.3 and 6.4 explicitly show how many runs are kept using each post selection scheme.

We note that in both cases, post-selection does not affect the soundness of the protocols. We only require that a non-negligible fraction of runs pass post-selection (to give good statistical significance for the results). This is indeed the case for our experiment, as can be seen in Tables 6.3, 6.4, as well as the statistical significance of the results in Tables 6.1, 6.2.

6.6.5 Shuttling and mid-circuit measurements

We control the position of the ions and run the split and shuttling sequences by changing the electrostatic trapping potential in a microfabricated chip trap [219] maintained at room temperature. We generate 40 time-dependent signals using a multi-channel DAC voltage source, which controls the voltages of the 38 inner electrodes at the center of the chip and the voltages of two additional outer electrodes. Owing to the strong radial confining potential used (with secular trapping frequencies near 3 MHz), the central electrodes' potential affects predominantly the axial trapping potential, and in turn, generates movement predominantly along the linear trap axis. To maintain the ions at a constant height above the trap surface, we simulate the electric field based on the model in Ref. [219], and compensate for the average variation of its perpendicular component by controlling the voltages of the outer two electrodes.

In the first sequence, we split the 15-ion chain into two sub-chains of 7 and 8 ions, and shuttle the 8-ion group to $x = 0.55$ mm away from the trap center at $x = 0$. We then align the 7-ion chain with the individual-addressing Raman beams for the first mid-circuit measurement. For the LWE-based protocol, we then reverse the shuttling process and re-merge the ions to a 15-ion chain, completing the circuit and performing a final measurement. For the factoring-based protocol, we shuttle the 8-ion sub-chain to the trap center and the 7-ion sub-chain to $x = -0.55$ mm. We then split this chain into 5- and 3-ion sub-chains, shuttle the 3-ion sub-chain to $x = 0.55$ mm, and align the 5 ions at the trap center with the Raman beams to perform additional gates and a second mid-circuit measurement. Finally, we move away the measured ions and align the 3-ion group to the center of the trap to complete the protocol. Reversing the sequence then prepares the ions in their initial state. For each protocol, all branches use the same shuttling sequences but differ in the qubit assignment and the realized gates. The mid-circuit measurement duration was experimentally determined prior to the experiment by maximizing the average fidelity of a Ramsey experiment using single-qubit gates, approximately optimizing for the trade-off between efficient detection of each sub-chain and stray light decoherence.

To enable efficient performance of the split and shuttling sequences we numerically simulate the electrostatic potential and the motional modes of the ions that are realized in the sequences. We minimize heating of the axial motion from low-frequency electric-field noise by ensuring that the calculated lowest axial frequency does not go below 100 KHz. We also

minimize the frequency of ions loss due to collisions with background gas by maintaining a calculated trap depth of at least 20 meV for each of the sub-chains throughout the shuttling sequences. The simulations enable efficient alignment of the sub chains with the Raman beams, taking into account the variation of the potential induced by all electrodes.

We account and correct for various systematic effects and drifts which appear in the experiment. To eliminate the effect of systematic variation of the optical phases between the individual beams on the ions, we align each ion with the same individual beam throughout the protocol. Prior to the experiment, we run several calibration protocols which estimate the electrostatic potential at the center of the trap through a Taylor series representation up to the fourth order, estimating the dominant effect of stray electric fields on the precalculated potential. We then cancel the effect of these fields using the central electrodes during the alignment and split sequences, as these sequences are most sensitive to the exact shape of the actual electrostatic potential. Additionally, we routinely measure the common-mode drift of the individual addressing optical Raman beams along the linear axis of the trap and correct for them by automatic repositioning of the ions achieved by varying the potential.

During shuttling, the ions traverse an inhomogeneous magnetic field and consequently, each ion spin acquires a shuttling-induced phase $\phi_s^{(i)}$ which depends on its realized trajectory. We calibrate this by performing a Ramsey sequence in which each qubit is put in a superposition of $(|0\rangle_i + |1\rangle_i)/\sqrt{2}$ before shuttling, and after the shuttling $R_x^{(i)}(\pi/2)R_z^{(i)}(\phi)$ gates are applied, with ϕ scanned from 0 to 2π. Fitting the observed fringe for each ion enables estimation of the phases $\phi_s^{(i)}$, which are corrected in the protocols by application of the inverse operation $R_z^{(i)}(-\phi_s^{(i)})$ after shuttling.

6.6.6 Circuit construction of the factoring-based protocol

In this section, we describe the procedure for preparing the quantum superposition of all the claws in the factoring-based protocol, as shown in Fig. 6.3(a) of the main text.

This is achieved by generating

$$\sum_{0\le x\le N/2} \frac{1}{\sqrt{2^{N/2}}} |x\rangle \, |f(x) = x^2 \bmod N\rangle \, . \tag{6.2}$$

and then measuring the $y = |f(x)\rangle$ register. We calculate $f(x)$ using a unitary $U(x, y)$ to encode the function into the phase of the y register and applying a inverse quantum Fourier transform ($\mathrm{QFT}^\dagger$) to extract the result.

To start, we apply Hadamard gates to all qubits to prepare a uniform superposition of all the possible bit strings for the x and y registers:

$$\sum_{0\le x\le N/2, 0\le y\le N} \alpha \, |x\rangle \, |y\rangle \, , \tag{6.3}$$

where α is the normalization factor.

Next, we evolve the state with the unitary $U(x,y) = e^{2\pi i \frac{x^2 y}{N}}$. Since the phase has period 2π, the unitary is equivalent to $U(x,y) = e^{2\pi i \frac{x^2 y \bmod N}{N}}$. We now show how to efficiently implement $U(x,y) = e^{2\pi i \frac{x^2 y}{N}}$ on the ion trap quantum computer.

First, note the multiplication in the phase can be expressed as a sum of bit-wise multiplication

$$U(x,y) = \prod_{i,j,k} \exp\left(2\pi i \frac{2^{i+j+k}}{N} x_i x_j y_k\right). \tag{6.4}$$

This bit-wise multiplication can be expressed using Pauli operators:

$$\prod_{i,j,k} \exp\left(2\pi i \frac{2^{i+j+k-3}}{N}(1-\sigma_z^{(i)})(1-\sigma_z^{(j)})(1-\sigma_z^{(k)})\right). \tag{6.5}$$

We then organize the operators into three terms:

$$U(x,y) = \prod_{i,j,k} \exp\left(\alpha_{i,j,k}\sigma_z^{(i)}\sigma_z^{(j)}\sigma_z^{(k)}\right) \prod_{i,j} \exp\left(\beta_{i,j}\sigma_z^{(i)}\sigma_z^{(j)}\right) \prod_i \exp\left(\gamma_i \sigma_z^{(i)}\right). \tag{6.6}$$

We use α's, β's, and γ's to represent the phases generated by these terms, which can be calculated from Eq. 6.5. The third term contains single-qubit Z rotations that are implemented efficiently as software-phase advances. The ZZ interactions in the second term are implemented as XX gates sandwiched between single qubit rotations. The first term includes three-body ZZZ interactions, which can be decomposed using ZZ interactions using the following relation:

$$\exp(-\frac{\pi}{4} i \sigma_y^{(i)}\sigma_y^{(j)}) \exp(i\theta\sigma_x^{(j)}\sigma_x^{(k)}) \exp(i\frac{\pi}{4}\sigma_y^{(i)}\sigma_y^{(j)}) = \exp(-i\theta\sigma_y^{(i)}\sigma_z^{(j)}\sigma_x^{(k)}) \tag{6.7}$$

This decomposition enables efficient construction of the following cascade of ZZZ interactions:

$$\exp(-i\theta_1\sigma_y^{(a)}\sigma_z^{(b)}\sigma_x^{(1)}) \exp(-i\theta_2\sigma_y^{(a)}\sigma_z^{(b)}\sigma_x^{(2)}) \cdots \exp(-i\theta_n\sigma_y^{(a)}\sigma_z^{(b)}\sigma_x^{(n)}) = \tag{6.8}$$

$$\exp(-\pi/4 i \sigma_y^{(a)}\sigma_y^{(b)}) \exp(i\theta_1\sigma_x^{(b)}\sigma_x^{(1)}) \cdots \exp(i\theta_n\sigma_x^{(b)}\sigma_x^{(n)}) \exp(i\pi/4\sigma_y^{(a)}\sigma_y^{(b)}) \tag{6.9}$$

which are efficiently implemented using the native XX interaction and single-qubit rotations.

Using the decomposition above, we can implement the first term in Eq. 6.6 using the circuit shown in Fig. 6.4.

With this term implemented, we complete the construction of the full unitary $U(x,y)$. After applying the unitary, we obtain the state

$$\alpha \sum_{0\le x\le N/2} |x\rangle \sum_{0\le y\le N} e^{2\pi i \frac{x^2 y \bmod N}{N}} |y\rangle. \tag{6.10}$$

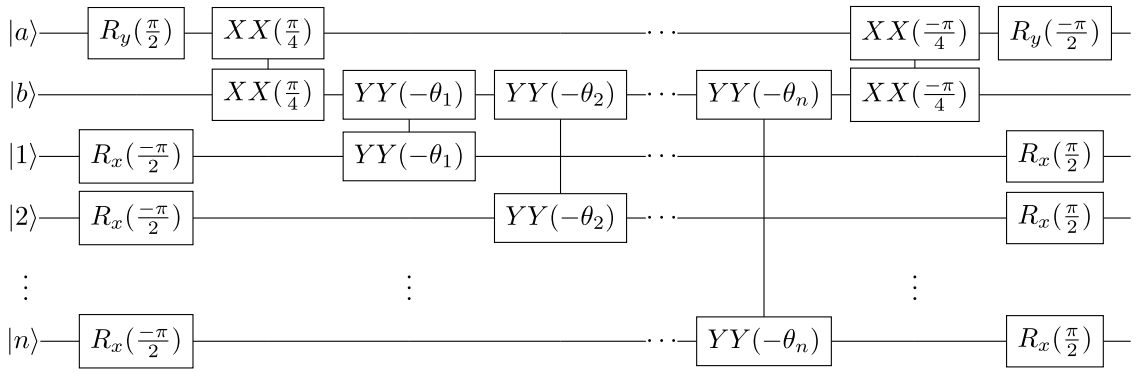


Figure 6.4: **Circuit compilation of products of three-body-Pauli interactions.** We use this optimized circuit to implemente the first term in Eq. 6.6.

We then apply the inverse quantum Fourier transform $QFT^\dagger$ to the y register, which gives us:

$$\alpha \sum_{0 \le x \le N/2} |x\rangle \, |y = x^2 \bmod N\rangle \tag{6.11}$$

Next, we measure the y register to find an output w, and the x-register contains the superposition of a colliding input pair.

The number of qubits used to represent y in experiments are 3, 4, 4, and 5 for $N = 8$, $N = 15$, $N = 16$, and $N = 21$, respectively. The number of qubits used to represent x in experiments equals the length of the r string in Table 6.4.

6.6.7 Circuit construction of the LWE-based protocol

In this section, we describe the procedure for implementing the circuit $U(A, b, x, y)$, displayed in Fig. 6.3(d). Here, the matrix A and vector y are classical inputs, and the bit b and vector x are quantum values held in the qubits to which the unitary is applied. The circuits evaluate the function $f(b, x) = \lfloor Ax + b \cdot y \rceil$ in superposition, where $\lfloor \cdot \rceil$ denotes a rounding operation corresponding to taking the most significant bit of each component in the vector $Ax+b\cdot y$. (It should be noted that this specific function, which uses rounding, differs from the TCF used by Brakerski et al. [39], but is nevertheless still a TCF [147].) This TCF is based on the LWE problem: for a secret vector s and "noise" vector e, find s (or equivalently e) with knowledge of only A and $y = As + e$. The LWE assumption states that doing so is cryptographically hard. It is straightforward to see that finding claws in the function f is as hard as breaking the LWE assumption, by observing that for a colliding pair $((0, x_0), (1, x_1))$, $x_1 = x_0 + s$ (up to the noise vector e, whose effect disappears due to the rounding).

In our implementation, the matrix $A \in \mathbb{Z}_q^{m \times n}$ and vector $s \in \{0, 1\}^n$ are sampled uni-

formly at random by the verifier[1]. The vector $e \in \mathbb{Z}_q^m$ is sampled from a discrete Gaussian distribution (see Brakerski et al. [39] for more details on the parameter choices). The verifier constructs $y = As + e \in \mathbb{Z}_q^m$ and sends A and y to the prover.

To perform the coherent evaluation, the prover will use three registers (for the b and x inputs, as well as for the output of the TCF) to create the superposition state as well as a fourth ancilla register, which will be used to perform the unitary $U(A, b, x, y)$. The prover starts by applying a layer of Hadamard gates to all input qubits and the ancilla register (that were initialized as $|0\rangle$). The resulting state will be

$$\sum_{b\in\{0,1\}} \sum_{x\in\mathbb{Z}_q^n} \sum_{a\in\mathbb{Z}_q} \alpha \,|b\rangle\,|x\rangle\,|a\rangle\,|0\rangle \tag{6.12}$$

for some normalization constant α and where the third register is the ancilla register and the last register is the output register. In this output register, the prover must coherently add $\lfloor Ax + b \cdot y \rceil$. As $Ax + b \cdot y$ is an m-component vector, we will explain the prover's operations, at a high level, for each component of the vector. For the i'th component of this vector, the prover first computes the inner product modulo q between the i'th row of A and x and places the result in the ancilla register. Since the prover has a classical description of A, this will involve a series of controlled operations between the x register and the ancilla register. Similar to the factoring case, this arithmetic operation is easiest to perform in the Fourier basis, which is why Hadamard gates are applied to the ancilla register. Once the inner product has been computed, the prover will perform a controlled operation between the b qubit and the ancilla register in order to add the i'th component of y. Finally, the prover will "copy" the most significant bit of the result into the output register. This is done via another controlled operation. The prover then uncomputes the result in the ancilla, clearing that register. In this way, the i'th component of $\lfloor Ax + b \cdot y \rceil$ has been added into the output register. Repeating this procedure for all components will yield the desired state

$$\sum_{b\in\{0,1\}} \sum_{x\in\mathbb{Z}_q^n} \alpha' \,|b\rangle\,|x\rangle\,|0\rangle\,|\lfloor Ax + b \cdot y \rceil\rangle \tag{6.13}$$

with normalization constant α'.

Having given the high level description, let us now discuss in more detail the specific circuits of the current implementation. From the above analysis, we can see that the total number of qubits is $N = 1 + n\log_2(q) + \log_2(q) + m$. In the instance for this experiment, we chose $m = 4, n = 2, q = 4$, resulting in $N = 11$ qubits. The first register contains $|b\rangle$ which requires only one qubit. In the second register, the vector $x = (x_0, x_1)$ consists of two components modulo 4, which is encoded in binary with four qubits as $|x\rangle = |x_{11}, x_{12}, x_{21}, x_{22}\rangle$. The third register, the ancilla, is one modulo 4 component and will thus consist of two qubits.

[1]Technically, the matrix A is sampled together with the TCF trapdoor. However, as explained in [39], the distribution from which the matrix is sampled is statistically close to a uniform distribution over $\mathbb{Z}_q^{m\times n}$

Lastly, in the fourth register, we store the result of evaluating the function, which requires another four qubits. As mentioned, the matrix A and the vector y are specified classically. In the experiment, we considered four different input configurations, corresponding to four different choices for A, s and e. These choices are explicitly described later in the appendix.

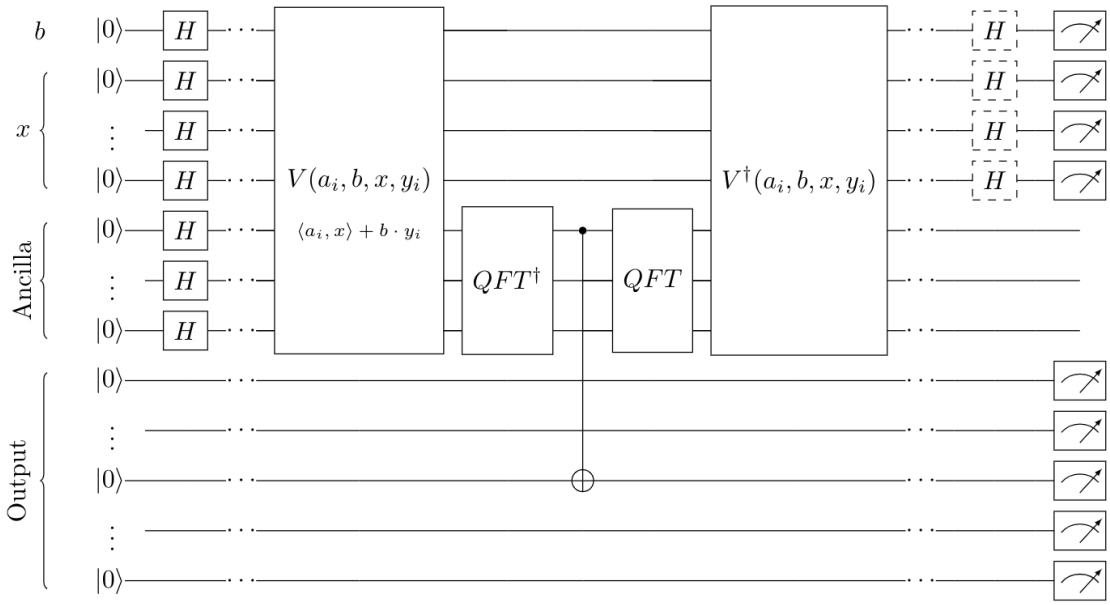


Figure 6.5: **Structure of circuits for the LWE protocols.** Circuit used to "copy" the most significant bit of the result from the ancilla into the output register, adding the i'th component of $\lfloor Ax + b \cdot y \rceil$. Here, V represents the unitary used to compute $\langle a_i, x\rangle + b \cdot y_i$, in modular arithmetic, for the i'th row a_i of the matrix A. Additionally, y_i denotes the i'th entry of the vector $y = As + e$. Also, note that the target qubit of the CNOT in the diagram is the i'th qubit. The step shown is repeated for each row of A, indexed by i.

To detail the operations implemented, as discussed previously, the prover first puts the ancilla register into the Fourier basis using the quantum Fourier transform (QFT). This allows them to more easily compute $\langle a_i, x\rangle + b \cdot y_i$ in the ancilla register, where a_i is the i'th row of the matrix A and $\langle \cdot, \cdot \rangle$ denotes the inner product modulo q. The explicit rotation gates to compute this in the Fourier basis are given in Fig. 6.6. After computing this for one row a_i, the prover converts the ancilla back into the computational basis and "copies" the most significant bit stored in the ancilla register into the output register, using a CNOT gate, to compute the rounding function. This completes the evaluation of the function for one bit. In order to reuse the qubits in the ancilla register, the prover then reverses this computation and repeats for each row of the matrix A. This process of evaluating the function and reversing that computation is depicted in Fig. 6.5.

Finally, after completing the evaluation of the TCF, the prover measures the output register to recover the rounded result of $\lfloor Ax + b \cdot y \rceil$ for a certain value of x. The prover will then measure the b and x registers in either the Z basis or X basis, according to the

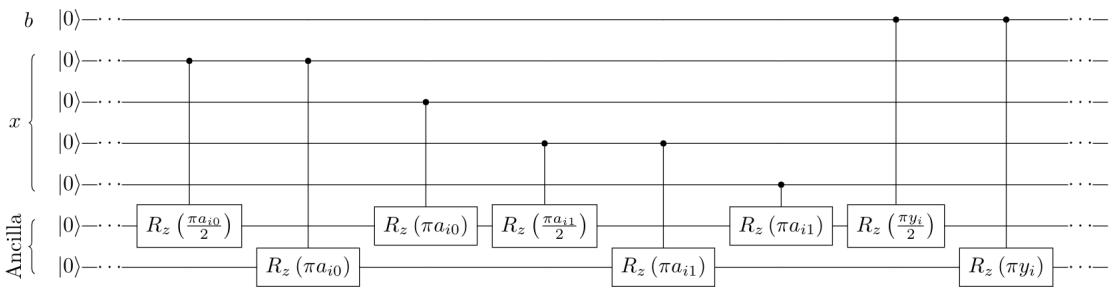


Figure 6.6: **Example of circuit for V from Fig. 6.5**. The explicit rotation gates used to implement the unitary V from Fig. 6.5 for the case of $q = 4$. Here, a_{ij} denotes the entry in the i'th row and j'th column of the matrix A and y_i denotes the i'th entry of the vector $y = As + e$. The output register is omitted as there are no operations performed on it in this section of the circuit. The step shown is repeated for each row of A, indexed by i.

challenge issued by the verifier. Should the verifier request measurements in the X basis, the prover applies Hadamard gates on all qubits in the b and x registers before measuring in the computational basis.

6.6.8 Instances of LWE Implemented

Instance	$A^\intercal$	$e^\intercal$	$(As+e)^\intercal$
0	$\begin{pmatrix} 0 & 2 & 0 & 1 \\ 2 & 0 & 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 3 & 0 & 1 \end{pmatrix}$
1	$\begin{pmatrix} 0 & 2 & 3 & 2 \\ 2 & 3 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 3 & 3 \end{pmatrix}$
2	$\begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & 3 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 3 & 0 & 1 & 1 \end{pmatrix}$
3	$\begin{pmatrix} 0 & 1 & 3 & 0 \\ 3 & 0 & 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 3 & 1 \end{pmatrix}$

Table 6.5: **Details of the LWE instances.** Note that the entries are transposed and for all instances we use $s^\intercal = \begin{pmatrix} 0 & 1 \end{pmatrix}$.

Here, we explicitly detail the LWE instances that were used in the experiment. Recall that such an instance is defined by A, s, and e, where $A \in \mathbb{Z}_q^{m \times n}$, $s \in \{0,1\}^n$, and $e \in \mathbb{Z}_q^m$ for integers $m, n, q \in \mathbb{Z}$. In this experiment, we used $m = 4, n = 2, q = 4$ for all of the instances. Table 6.5 displays the explicit matrices and vectors used.

6.6.9 Discussion of asymptotic classical threshold

In cryptography, showing that a new protocol is secure for practical use (meaning, in our case, that the proof cannot be spoofed by a classical prover) follows two broad steps: 1) proving that it is secure asymptotically (showing that the computational cost of cheating is at least superpolynomial in the problem size), and 2) picking a finite set of parameters such that cheating is not possible under certain classical resources (computational power and time, usually). What particular limitations to set on the resources available to the classical cheater is ultimately up to the user. In this section, we attempt to make precise exactly which statements are asymptotic (step 1), and how these statements make the jump in step 2 to finite, real parameters.

The first asymptotic statement, which is perhaps the most obvious, is that finding claws of the TCF is hard. In the theory papers upon which this work is based, this is shown by reducing the problem of finding claws to related problems, about which there exist standard cryptographic assumptions. [39], [115] In particular, the assumptions are that the factoring and LWE problems have superpolynomial classical complexity. As discussed above, when using the test in practice we would pick finite parameters in a way that finding a claw is infeasible for the set of classical resources we wish our quantum computer to outcompete (for a rigorous demonstration of quantum advantage, probably a large supercomputer with ample runtime). Importantly, the reduction between the hardness of finding claws and breaking the cryptographic assumption is not in any sense asymptotic: for both TCFs, if a machine can find claws for a specific, finite set of parameters, they can directly use those claws to break the cryptographic assumption in practice. Therefore if the cryptographic assumption holds for a finite set of parameters, we can be sure that the claw-freeness does as well.

The second asymptotic statement that appears in the analysis of these protocols is in regard to the probability that a classical cheater passes a single iteration of the test. In Section 6.4 of the main text, we discuss the "classical thresholds," which must be exceeded to demonstrate quantum capability. To be very precise about what we mean by this, we reproduce exactly what the theorems underlying these protocols state: if a classical prover's true success probabilities (not the empirically determined ones, which are subject to statistical fluctuation) exceed the given bound by a non-negligible amount, that prover could be used as a subroutine in a larger program which finds a claw in the TCF in polynomial time. Thus, if it is not possible for a classical prover with certain resources to find a claw (in a TCF with some specific parameters), it is provably also not possible for a classical prover with similar resources to non-negligibly exceed the threshold. There are two asymptotic portions of this statement: the polynomial time in which the larger program extracts a claw using the prover as a subroutine (which is the reason for the word "similar" in the previous sentence), and in fact the word "negligible." Negligible has a technical definition in cryptography, which is the sense in which we use it here. It means that a value (in this case the amount by which the threshold can be exceeded) is bounded by a function which goes exponentially to zero in the problem size. The precise form of this exponential is not intended to be determined, but instead the exponential decay is used to argue that the negligible function is "essentially"

zero for any reasonable problem size that would be used in practice.

It is worth noting that for the small problem sizes we implement in this work, there is one instance in which this negligible function would meaningfully affect the classical success threshold—and we modify the protocol slightly to account for this. In the x^2 mod N (Rabin's function) protocol, the value r sent by the verifier is supposed to be a uniformly random bitstring. If r happens to be all zero, the product $r \cdot x$, whose value is supposed to be cryptographically hard to guess, is simply zero. This is not an issue for problem sizes that would be used for a full-scale test in practice, because an all-zero r is extremely unlikely to occur if r is of length several hundred bits. But for our smaller experiments with r of only a few bits, the all-zero string represents a sizable fraction of possible r. To prevent this from affecting the results, we simply choose our r from the set of non-zero bitstrings rather than all bitstrings. We note that excluding the all zero string helps us better resolve the experiment's performance, too: when $r = 0^n$ the qubit measured in the last step of the protocol never interacts with any of the other qubits throughout the whole circuit, so the measurement result has nothing to do with the fidelity of the TCF circuit!

To close this discussion, it is worth taking a broader perspective and considering how the field of cryptography functions in general. Asymptotic proofs in cryptography are used to show that for any cheating machine with finite resources, the problem can always be made large enough to be hard in practice—and that the hardness grows quickly enough that this is hopefully not an unreasonable pursuit. But ultimately, the question of how large the problem needs to be is an empirical one: experts build the best possible algorithms and hardware they can to attempt to break the assumption, and then the parameters are set to be larger than the largest problem size that can be broken this way (usually with an extra buffer added to secure against improvements in the attacks). In our case, the costs of breaking both factoring and LWE have been extensively explored, and the practical parameters needed for their security against current classical computing power are well understood. As described above, because there are no asymptotic statements in the reduction from the TCF to the underlying cryptographic assumptions, these parameters can be directly used to ensure that finding claws is hard in practice. As described above, the precise relationship between the hardness of exceeding the thresholds and finding claws does rely on asymptotics, but the fact that the asymptotic function appearing in the threshold is shown to decay exponentially suggests strongly that this should not be an issue in practice.

6.6.10 Quantum-classical threshold of LWE based protocol

In this section, we state and prove the classical threshold for the LWE-based protocol. The corresponding proof for the factoring-based protocol is contained in the theory manuscript that first presented that protocol. [115]

Below, the security parameter λ is used in the standard cryptographic sense, as a measure of the "problem size"—it can be made larger to increase security, or smaller to improve efficiency. The specifics of how each parameter of the LWE problem is defined as a function

of λ can be found in the definition of the LWE-based TCF, in the theory work that originally proposed it. [39]

Proposition 1. *For any classical prover, the probabilities that they pass branches A and B, p_A and p_B, must obey the relation*

$$p_A + 2p_B - 2 < \epsilon(\lambda) \tag{6.14}$$

where ϵ is a negligible function of the security parameter λ.

Proof. We first want to find the probability that the classical prover both responds correctly for Branch A and, for the same output w that they committed to the verifier, Branch B is also correct with probability greater than $1/2 + \mu(\lambda)$, where μ is a non-negligible function of the security parameter λ. Let this second probability be denoted as

$$p_{\text{good}} \equiv \Pr_w[p_{B,w} > 1/2 + \mu(\lambda)] \tag{6.15}$$

By a union bound, we arrive at a bound on the desired probability

$$\Pr[A \text{ correct and } p_{B,w} > 1/2 + \mu(\lambda)] > p_A + p_{\text{good}} - 1 \tag{6.16}$$

Now, we wish to write p_{good} in terms of p_B. Let S be the set of w values for which $p_{B,w} > 1/2 + \mu(\lambda)$. By definition, we know that with probability p_{good}, the prover samples a $w \in S$ so that they pass the verifier's Branch B test with probability at least $1/2 + \mu(\lambda)$ and at most 1. Similarly, we know that with probability $1 - p_{\text{good}}$, the prover samples a $w \notin S$ so that they pass the verifier's Branch B test with probability at most $1/2$. Hence, overall we see that the probability that the prover passes Branch B is at most the convex mixture of these two cases, i.e.

$$p_B < 1 \cdot p_{\text{good}} + 0.5 \cdot (1 - p_{\text{good}}) \tag{6.17}$$

Solving for p_{good}, we then obtain

$$p_{\text{good}} > 2p_B - 1 \tag{6.18}$$

Substituting this into Equation 6.16, we have

$$\Pr[A \text{ correct and } p_{B,w} > 1/2 + \mu(\lambda)] > p_A + 2p_B - 2 \tag{6.19}$$

However, notice that this probability on the left hand side is the probability of breaking the adapative hardcore bit property, which we know [39] must have

$$\Pr[A \text{ correct and } p_{B,w} > 1/2 + \mu(\lambda)] < \epsilon(\lambda) \tag{6.20}$$

where ϵ is a negligible function. Thus, combining this with Equation 6.19, we obtain the desired inequality

$$p_A + 2p_B - 2 < \epsilon(\lambda) \tag{6.21}$$

□

6.6.11 Computation of statistical significance contours

Here we describe the computation of the contour lines denoting various levels of statistical significance in Figure 6.3(b,e) of the main text. Recall the probabilities p_A and p_B introduced in Section 6.4, which denote a prover's probability of passing the standard basis and interference test, respectively. Assuming the cryptographic soundness of the claw-free property of the TCF, and in the limit of large problem size, any classical cheating strategy must have true values of p_A^c and p_B^c that obey the bound $p_A^c + 2p_B^c - 2 < 0$ for the LWE protocol and $p_A^c + 4p_B^c - 4 < 0$ for the factoring-based protocol. To find the statistical significance of a pair of values p_A and p_B measured from an (ostensibly) quantum prover, we consider the null hypothesis that the data was generated by a classical cheater (which obeys the bounds above), and compute the probability that the given data could be generated by that null hypothesis. In particular, since the bounds above exclude a region of a two-dimensional space, we consider an infinite "family" of null hypotheses which lie along the boundary, and define the overall statistical significance of measuring p_A and p_B to be the minimum of the statistical significances across the entire family of null hypotheses—that is, we define it as the significance with respect to the least rejected null hypothesis.

To compute the statistical significance of a result (p_A, p_B) with respect to a particular null hypothesis (p_A^c, p_B^c), we define the "quantumness" q of an experiment as $q(p_A, p_B) = p_A + 4p_B - 4$ for the factoring-based protocol and $q(p_A, p_B) = p_A + 2p_B - 2$ for the LWE protocol. Letting N_A and N_B be the number of experimental runs performed for each branch respectively, we define the joint probability mass function (PMF) as the product of the PMFs of two binomial distributions $B(N_A, p_A^c)$ and $B(N_B, p_B^c)$. Mathematically the joint PMF is

$$f(k_A, k_B; p_A^c, p_B^c, N_A, N_B) = \binom{N_A}{k_A}\binom{N_B}{k_B}(p_A^c)^{k_A}(p_B^c)^{k_B}(1-p_A^c)^{N_A-k_A}(1-p_B^c)^{N_B-k_B} \quad (6.22)$$

where $k_A = p_A N_A$ and $k_B = p_B N_B$ are the "count" of passing runs for each branch respectively. Finally, we compute the statistical significance of a result (p_A, p_B) as the probability of achieving quantumness measure of at least $q' = q(p_A, p_B)$. Under a null hypothesis (p_A^c, p_B^c), this is the sum of the PMF over all k_A, k_B for which $q(k_A/N_A, k_B/N_B) > q'$.

In practice, for the contour lines of Figure 6.3(b,e), we begin with a desired level of statistical significance (say, 5σ), and given the sample sizes N_A and N_B we compute the value of q' that would achieve at least that significance over all null hypotheses inside the classical bound.

6.6.12 Additional instances of factoring-based protocols

In Fig. 6.3(f), we show the relative performance of the factoring-based protocol for $N = 8$, performed both interactively and with delayed measurement. In Fig. 6.7 we display the relative performance for $N \in \{15, 16, 21\}$ (for which experiments were run with delayed measurement only).

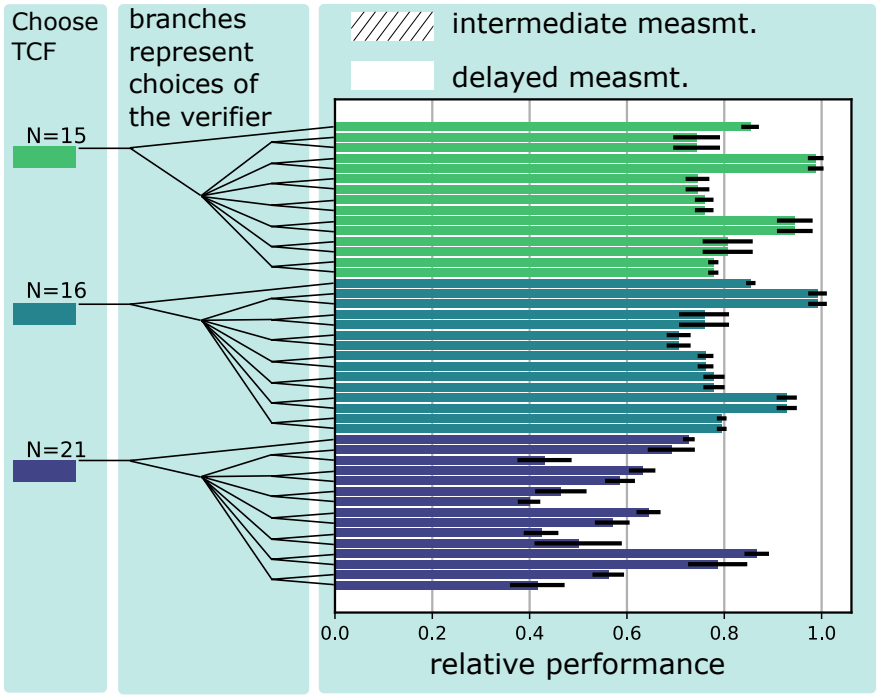


Figure 6.7: **Extra implementations of the factoring-based protocols.** Other than the implementations of N=8 instances, we also experimented with N=15,16, and 21 instances of the factoring-based protocols, with delayed measurement.

6.6.13 Estimate of resources required to achieve quantum advantage

For a conclusive demonstration of quantum advantage, we desire the quantum machine to perform the protocol significantly faster than the amount of time a classical supercomputer would require to break the trapdoor claw-free function—ideally, orders of magnitude faster. To achieve this, we must set the parameters of the cryptographic problem to sufficiently large values. A major benefit of using protocols based off of established cryptographic assumptions (like factoring and LWE) is that the classical hardness of breaking these assumptions is extremely well-studied, due to the implications for security. [136] Thus the most straightforward way to choose parameters for our tests is to rely on publicly-available recommendations for cryptographically secure key sizes, which are used in practice. These parameter settings are designed to be not just slow for classical machines, but infeasible even for classical machines years from now—and thus certainly would constitute a definitive demonstration of quantum advantage. However, setting the parameters to these values may be considered overkill for our purposes, especially since we'd like the problem size to be as small as possible in order to make the protocols maximally feasible on near term quantum devices. With these considerations, in this section we provide two estimates for each protocol: we begin by providing estimates for smaller problem sizes that still would demonstrate some level of

quantum advantage, and then give estimates based on cryptographic parameters.

We conservatively estimate that a future quantum device running the protocols investigated in this work at scale would complete the protocols on a time scale of at most hours. Thus, to demonstrate quantum advantage by several orders of magnitude, we desire to set the parameters such that a classical supercomputer would require time on the order of thousands of hours to break the TCF. In 2020, Boudot et al. reported the record-breaking factorization of a 795-bit semiprime [220]. The cost of the computation was about 1000 core-years, meaning that a 1000-core cluster would complete it in a year. We consider this sufficient cost to demonstrate quantum advantage. We emphasize also that factoring is one of the most well-studied hard computational problems; the record of Boudot et al. is the product of decades of algorithm development and optimization and thus it is unlikely that any innovations will drastically affect the classical hardness of factoring in the near term. The computational Bell test protocol using a 795-bit prime could be performed using only about 800 qubits by computing and measuring the bits of the output value w one-by-one; however the gate count and circuit depth can be dramatically reduced by explicitly storing the full output value w, requiring roughly 1600 qubits total [115]. Because it is so much more efficient in gate count, we use the 1600 qubit estimate as the space requirement to demonstrate quantum advantage with the computational Bell test protocol.

For LWE, estimating parameters for the same level of hardness (1000 core-years) is difficult to do exactly, because to our knowledge that amount of computational resources has never been applied to breaking an LWE instance. However, we may make a rough estimate. There is an online challenge (`https://www.latticechallenge.org/lwe_challenge/challenge.php`) intended to explore the practical classical hardness of LWE, in which users compete for who can break the largest possible instance. As of this writing, the largest instances which have been solved use LWE vectors of about 500-1000 bits (depending on the noise level of the error vector), but the computational cost of these calculations was only of order 0.5 core-years. To require 1000 core-years of computation time, we estimate that the LWE vectors would need to be perhaps 1000-2000 bits in length; by not explicitly storing the output vector w but computing it element-by-element (similar in principle to the scheme for evaluating $x^2 \bmod N$ using only $\log(N) + 1$ qubits [115]) it may be possible to perform the LWE protocol using a comparable number of qubits to the bit length of one LWE vector.

We now provide estimates for cryptographic parameters; that is, parameters for which it expected to be completely infeasible for a classical machine to break the trapdoor claw-free function. For the factoring-based protocol, we may apply NIST's recommended key sizes for the RSA cryptosystem, whose security relies on integer factorization. NIST recommends choosing a modulus N with length 2048 bits. By using circuits optimized to conserve qubits, it is possible to evaluate the function $x^2 \bmod N$ using only $\log(N) + 1$ qubits, yielding a total qubit requirement of 2049 qubits [115]. However, the circuit depth can be improved significantly by including more qubits; a more efficient circuit can be achieved with roughly $2\log(N) \sim 4100$ qubits. Because LWE is not yet broadly used in practice like RSA is, NIST does not provide recommendations for key sizes in its documentation. However, we can use the estimates of Lindner and Peikert[221] to find parameters which are expected to be

infeasible classically. In Fig. 3 of that work, the authors suggest using LWE vectors in $\mathbb{Z}_q^n$ with $n = 256$ and $q = 4093$ for a "medium" level of security. Vectors with these parameters are $n \log(q) \sim 3072$ bits long. To store both an input and output vector would thus require roughly ~ 6200 qubits. By repeatedly reusing a set of qubits to compute the output vector element-by-element the computation could be performed using roughly 3100 qubits.

Chapter 7

Quantum fast multiplication with very few qubits

7.1 Introduction

Quantum circuits for coherently performing arithmetic on superpositions of values have been a subject of intense study since the first quantum algorithms for number theory problems were discovered in the mid-1990s. The most famous such algorithms are Shor's algorithms for integer factorization and discrete logarithm, because of their potentially catastrophic effect on digital security. [119] Among other applications, quantum arithmetic has recently become useful also for the implementation of protocols to achieve various quantum cryptographic tasks—from "proofs of quantumness" which allow a quantum device to demonstrate its computational capability to a skeptical classical verifier, to practical applications like secure delegation of computations to an untrusted remote quantum server. [39], [41], [47]–[49], [115]

The standard way of performing multiplication (both in the classical and quantum setting) is via the "schoolbook" algorithm, that uses $\mathcal{O}(n^2)$ gates, where n is the size of the input. While the existence of faster classical algorithms for multiplication has been known for over a half-century, these algorithms have overheads that make them only useful for multiplication of large values—the GNU multiple-precision arithmetic library uses a threshold of 2176 bit inputs to switch away from the schoolbook method. In the quantum setting, these overheads have generally proven to be made even worse by the reversibility constraints of quantum circuits. Despite these challenges, several works have explored and optimized the implementation of quantum circuits for sub-quadratic time multiplication, largely focusing on the Karatsuba algorithm which has an asymptotic runtime of roughly $\mathcal{O}(n^{1.58})$. [176], [183], [184], [222], [223] For many years, a significant challenge came from the fact that these fast algorithms are recursive—and building reversible circuits for them while maintaining the speedup required storing intermediate data, ultimately using a superlinear number of qubits. Notable is a recent work which for the first time reduced the number of qubits for

quantum Karatsuba multiplication to linear in the size of the inputs, by performing extra manipulations on the output register that enable a strategy akin to *tail recursion*, where intermediate results are summed directly into the output and do not need to be uncomputed later. [176] That work provides hope for very efficient quantum integer multiplication, and encourages the search for improvement in the gate count and especially the qubit count (multiplying 2048 bit numbers with that algorithm still requires over 10,000 ancilla qubits).

In this work, we explore a new paradigm for the design of sub-quadratic quantum multiplication circuits. We show that by combining the fundamental ideas behind fast multiplication algorithms with an inherently quantum technique where arithmetic is performed in the phases of a quantum state, it is possible to design circuits for quantum multiplication that simultaneously achieve sub-quadratic asymptotic gate counts and a *constant* (and small) number of ancilla qubits: just two ancillas for multiplication of a quantum value by a classical one, and five ancillas for the multiplication of two quantum values. Our results yield a family of algorithms with varying tradeoffs between constant factors and asymptotic complexity; by estimating the gate counts for our circuits we find that for 2048-bit inputs, our $\mathcal{O}(n^{1.46})$ algorithm is the optimal one for multiplication of a quantum integer by a classical one, and our $\mathcal{O}(n^{1.66})$ one is optimal for multiplying two quantum integers. We also find that our algorithms begin to outperform the schoolbook algorithm in gate count quite early—for inputs with fewer than 100 bits in some cases. In terms of concrete comparisons of circuit sizes (which we note are difficult to make fairly in the abstract circuit model) we find that our algorithms dramatically reduce the number of qubits required for subquadratic quantum multiplication, while simultaneously achieving competitive gate counts. We also note that our circuits do not require intermediate measurements for measurement-based uncomputation (indeed, there are so few garbage bits produced that there is essentially nothing to uncompute!). In terms of depth, we find that surprisingly, we are limited to $\mathcal{O}(n)$ depth only because we do not know of a way to perform the quantum Fourier transform in less than $\mathcal{O}(n)$ depth with such limited space—even if we allow the use of $\mathcal{O}(n)$ ancillas and for the QFT to be approximate. On the other hand, the core "phase arithmetic" portion of our algorithm can be performed in sub-linear depth using $\mathcal{O}(n)$ ancillas.

Directly using our circuits as a subroutine in Shor's algorithm yields circuits that require only $2n+\mathcal{O}(\log n)$ qubits, where n is the length of the integer to be factored—while obtaining an asymptotic gate count of $\mathcal{O}(n^{2.46})$ with practical constant factors. This is compared to the standard $\mathcal{O}(n^3)$ gate complexity of Shor's algorithm.[1] We also explore the circuits' application to the efficiently-verifiable cryptographic proof of quantumness introduced in Chapter 5; we see dramatic reductions in the qubit count along with competitive gate counts, when compared to previous implementation proposals. The new circuits also do not require the use of measurement-based uncomputation.

[1] It has been known for many years that it is theoretically possible to reduce the asymptotic complexity of Shor's algorithm below $\mathcal{O}(n^3)$ [224], but previously the constant factors and number of qubits required seemed to make doing so impractical for reasonably sizes inputs. See Table 1 of [177] for a recent comparison of proposals for the implementation of Shor's algorithm.

Throughout this manuscript we work in the abstract circuit model of quantum computation; a critical direction for future research is how to optimize our constructions to include considerations such as error correction, qubit routing, and noise.

7.2 Background

7.2.1 Quantum multiplication

In this work we will focus on implementing two related quantum arithmetic operations. The first is quantum-classical multiplication: computing the product of (a superposition of) integers stored in a quantum register with a classical integer a. We denote the unitary corresponding to this operation as $\mathcal{U}_{q\times c}(a)$ (the subscript denotes "quantum" $\times$ "classical", and the unitary is parameterized by a). It implements the following operation on product states (extended by linearity to superpositions):

$$\mathcal{U}_{q\times c}(a)\,|x\rangle\,|w\rangle = |x\rangle\,|w + ax\rangle \tag{7.1}$$

We note that this unitary (controlled off of another qubit) is the fundamental operation used in implementations of Shor's algorithm (see Section 7.4.3).

The second operation we study is a quantum-quantum multiplication: finding the product of an integer (or superposition thereof) in one quantum register with another integer (or superposition) in a second register. We denote this as $\mathcal{U}_{q\times q}$; it implements the transformation

$$\mathcal{U}_{q\times q}\,|x\rangle\,|y\rangle\,|w\rangle = |x\rangle\,|y\rangle\,|w + xy\rangle \tag{7.2}$$

In Sec. 7.4.4 we also discuss the closely related unitary for which $|x\rangle$ and $|y\rangle$ of the previous equation are the same register, thus implementing a squaring operation:

$$\mathcal{U}_{\text{square}}\,|x\rangle\,|w\rangle = |x\rangle\,|w + x^2\rangle \tag{7.3}$$

This squaring operation is fundamental to implementing "proofs of quantumness" based on the cryptographic function $f(x) = x^2 \bmod N$, as discussed in Chapter 5.

7.2.2 Fast multiplication algorithms

We now discuss classical multiplication algorithms. The most straightforward algorithm for multiplying two numbers is known as the "schoolbook" method, because it is the one taught to young students when they first learn to multiply. The numbers are simply decomposed into individual digits and the individual products of those digits summed, scaled by powers of the base. In binary this can be expressed as

$$xy = \sum_{i,j} 2^{i+j} x_i y_j \tag{7.4}$$

(presumably most elementary school teachers use base 10). This algorithm's time complexity is $\mathcal{O}(n^2)$; it is used widely for multiplication of small- to moderate-sized integers.

About half a century ago it was shown that it is possible to classically multiply integers in sub-quadratic time—asymptotically outperforming the schoolbook algorithm. We begin by describing the first sub-quadratic multiplication algorithm, called the Karatsuba algorithm, and then show that it is a special case of a broader class of algorithms called Toom-Cook. For an extended pedagogical exposition of the range of fast multiplication algorithms that have been discovered, we refer the reader to Knuth. [225]

7.2.2.1 Karatsuba multiplication

Consider two n-bit integers x and y to be multiplied together. Divide the bits of each into two pieces: $x = 2^{n/2}x_1 + x_0$ (and the same for y), where x_1 is the bits of the "more significant" half and x_0 is the "less significant" one. Written in this way, the product can be expressed

$$xy = (2^{n/2}x_1 + x_0)(2^{n/2}y_1 + y_0) = 2^n x_1 y_1 + 2^{n/2}(x_0 y_1 + x_1 y_0) + x_0 y_0 \tag{7.5}$$

This formulation is effectively the schoolbook algorithm in base 2^n. We've replaced one product of size n with four products of size $n/2$, reflecting the quadratic scaling of the schoolbook algorithm.

The key observation behind the Karatsuba algorithm is that

$$x_0 y_1 + x_1 y_0 = (x_0 + x_1)(y_0 + y_1) - x_0 y_0 - x_1 y_1 \tag{7.6}$$

—and we already need to find the products $x_0 y_0$ and $x_1 y_1$ anyway! Making this substitution we have

$$xy = 2^n x_1 y_1 + 2^{n/2}((x_0 + x_1)(y_0 + y_1) - x_0 y_0 - x_1 y_1) + x_0 y_0 \tag{7.7}$$

That is, the multiplication of size n can be accomplished with only *three* multiplications of size $n/2$, and a few extra additions! The next key insight is to apply this fact recursively, using it again to compute each of the sub-multiplications of size $n/2$, and then again to compute the multiplications used there, et cetera. Complexity analysis shows that when applied recursively, Karatsuba multiplication computes the product in only $\mathcal{O}(n^{\log_2 3}) = \mathcal{O}(n^{1.58\cdots})$ operations, outperforming the schoolbook algorithm.

7.2.2.2 Toom-Cook multiplication

The Toom-Cook algorithm uses the same intuition as Karatsuba, but splits each integer into k parts instead of just two. It also provides a broader intuition for why the Karatsuba algorithm works, by expressing the problem in terms of *polynomial multiplication*.

Consider an n-digit integer x divided into k chunks of size n/k, which we denote $x_0, x_1, \cdots, x_{k-1}$ with x_{k-1} being the most significant:

$$x = x_{k-1}2^{(k-1)n/k} + \cdots + x_1 2^{n/k} + x_0 \tag{7.8}$$

This can be reinterpreted as a polynomial

$$x(w) = x_{k-1}w^{k-1} + \cdots + x_1 w + x_0 \tag{7.9}$$

evaluated at $w = 2^{n/k}$. From this perspective, we can recast a product of integers $p = xy$ as a product of polynomials $p(w) = x(w)y(w)$, evaluated at $w = 2^{n/k}$. The benefit of expressing the product this way is that for *any* point w_i, $p(w_i) = x(w_i)y(w_i)$, and we can choose values of w_i such that $x(w_i)$ and $y(w_i)$ are only roughly n/k bits long and thus the product is faster to compute. This suggests the following plan: compute the value of $p(w)$ at several points, use those points to reconstruct the coefficients of the polynomial $p(w)$, and finally evaluate $p(w)$ at $w = 2^{n/k}$ yielding the integer product z. $x(w)$ and $y(w)$ are polynomials of degree $k-1$, so their product $p(w)$ is of degree $2(k-1)$, thus we must evaluate it at $q = 2(k-1)+1$ points to unambiguously reconstruct the coefficients.

It may be helpful at this point to recast the Karatsuba algorithm in this light. Karatsuba corresponds to the case of $k = 2$; the integer x is converted into the polynomial $x(w) = x_1 w + x_0$ (and similarly for y). Here $q = 2(k-1)+1 = 3$, and so we evaluate the product $p(w_i) = x(w_i)y(w_i)$ at the three points $w_i \in \{0, 1, \infty\}$:

$$p(0) = x(0)y(0) = x_0 y_0 \tag{7.10}$$
$$p(1) = x(1)y(1) = (x_0 + x_1)(y_0 + y_1) \tag{7.11}$$
$$p(\infty) = x(\infty)y(\infty) = x_1 y_1 \tag{7.12}$$

(where the value of polynomial p at infinity is the limit of $p(w)/w^{\deg(p)}$ as $w \to \infty$). The coefficients of the polynomial $p(w)$ can be expressed in terms of its value at these three points as follows:

$$p(w) = p(\infty)w^2 + (p(1) - p(\infty) - p(0))w + p(0). \tag{7.13}$$

Substituting Eqns. 7.10-7.12 into Eq. 7.13 and evaluating at $w = 2^{n/2}$ yields Eq. 7.7, the expression we originally had for computing p in Karatsuba multiplication!

The benefit of framing our multiplication in this way is that it becomes very straightforward to set k to values larger than 2. The Toom-Cook algorithm can be viewed as a five-step process, which is conveniently expressed in terms of linear algebra: [226]

1. Split We divide the bits of x and y into k chunks each, which serve as the coefficients of a polynomial as in Eq. 7.9 above. We can represent this polynomial simply as a vector of the coefficients:

$$\boldsymbol{x} = (x_{k-1}, \cdots, x_1, x_0) \tag{7.14}$$

In this representation, evaluating the polynomial at a point w_i corresponds to computing the inner product $\boldsymbol{e}^\intercal \boldsymbol{x}$ with a vector

$$\boldsymbol{e}(w_i) = (w_i^{k-1}, \cdots, w_i, 1) \tag{7.15}$$

2. Evaluate The next step is to evaluate those polynomials at several points. We define a length $q = 2(k-1)+1$ vector $\tilde{\boldsymbol{x}}$ which contains the evaluation of our polynomial at each of q points $w_0, w_1, \cdots$. With this we write $\tilde{\boldsymbol{x}} = A\boldsymbol{x}$, where each row of the matrix A is $\boldsymbol{e}(w_i)$ for one of the points w_i at which we desire to evaluate the polynomial. Later it will be helpful to write A as a $q \times q$ matrix; we'll allow multiplication by the length k vector $\boldsymbol{x}$ by simply extending prepending zero entries to $\boldsymbol{x}$ for the higher order terms. With that, A has the following structure:

$$A = \begin{pmatrix} w_0^{q-1} & \cdots & w_0^2 & w_0 & 1 \\ w_1^{q-1} & \cdots & w_1^2 & w_1 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ w_{q-1}^{q-1} & \cdots & w_{q-1}^2 & w_{q-1} & 1 \end{pmatrix} \tag{7.16}$$

We perform that matrix-vector multiplication for both $\boldsymbol{x}$ and $\boldsymbol{y}$.

3. Multiply Now that x and y are expressed as polynomials evaluated at several points, we reach the payoff: we multiply the polynomials by simply performing pointwise multiplication of the vectors:

$$\tilde{\boldsymbol{p}} = \tilde{\boldsymbol{x}} \circ \tilde{\boldsymbol{y}} \tag{7.17}$$

where $\tilde{\boldsymbol{p}}$ is the result polynomial evaluated at the points w_i, and $\circ$ denotes the pointwise product. Note the benefit here: we are performing $2q$ pointwise products of size n/k, instead of the k^2 products of that size that would be required for schoolbook multiplication. Since $q = 2(k-1)+1$ this is a very dramatic improvement! Furthermore we will recursively apply the Toom-Cook algorithm to each of the pointwise multiplications of the vector elements.

4. Interpolate Now that we have the vector $\tilde{\boldsymbol{p}}$ representing the product polynomial evaluated at each of the q points w_i, we need to convert it to a vector of polynomial coefficients $\boldsymbol{p}$. We know that $\tilde{\boldsymbol{p}} = A\boldsymbol{p}$; by appropriate choice of the w_i we can ensure that A is invertible. Then we can simply compute the matrix inverse A^{-1}, resulting in a straightforward computation $\boldsymbol{p} = A^{-1}\tilde{\boldsymbol{p}}$.

5. Recomposition Finally, with the product polynomial expressed in terms of its coefficients $\boldsymbol{p}$, the integer product can be recovered by evaluating that polynomial at $w = 2^{n/k}$. As before this can be written as the inner product $(\boldsymbol{e}(2^{n/k}))^\intercal \boldsymbol{p}$, where

$$\boldsymbol{e}(2^{n/k}) = (2^{(q-1)n/k}, \cdots, 2^{2n/k}, 2^{n/k}, 1) \tag{7.18}$$

With the algorithm laid out, let us the asymptotic complexity of Toom-Cook for any k. The algorithm performs $q = 2(k-1)+1$ multiplications of size n/k at the top level, and then recurses. Straightforward complexity analysis yields a runtime of $\mathcal{O}(n^{\log_k q})$, which can be made arbitrarily close to $\mathcal{O}(n)$ as k is increased. Unfortunately in practice the constant factors grow rapidly as k is increased, due to the cost of performing the matrix vector

products $A\boldsymbol{x}$, $A\boldsymbol{y}$ and $(A^{-1})\boldsymbol{p}$. In practice, the value of k is varied depending on the size of the integers to be multiplied, but regardless of the size of input k is only practicable for small values (certainly less than 10). [227]

7.2.3 Fourier arithmetic

Many proposals for the implementation of quantum arithmetic mirror classical circuits, with various clever optimizations to account for the fact that quantum operations must in general be reversible. [176], [181], [228]–[232] However, one early work by Draper stands out as an example of a quantum arithmetic circuit that truly has no classical analogue. [156] That work proposes to use the quantum Fourier transform (QFT) to compute addition, and to use that addition circuit as a building block for implementing multiplication.

The action of the quantum Fourier transform, with modulus 2^m, on a product state $|x\rangle$ of m qubits can be written as follows (and by linearity extended to a superposition of inputs):

$$\mathsf{QFT}_{2^m} |x\rangle = \sum_z \exp\left(2\pi i \frac{xz}{2^m}\right) |z\rangle \tag{7.19}$$

(and the inverse quantum Fourier transform IQFT_{2^m} is the reverse of this operation). Draper observed that one can perform addition in the Fourier basis simply by applying a series of phase rotations to the qubits of the register holding z:

$$\mathsf{QFT}_{2^m} |x+a\rangle = \sum_z \exp\left(2\pi i \frac{(x+a)z}{2^m}\right) |z\rangle = \sum_z \exp\left(2\pi i \frac{az}{2^m}\right) \exp\left(2\pi i \frac{xz}{2^m}\right) |z\rangle \tag{7.20}$$

The rightmost expression is just $\mathsf{QFT}_{2^m} |x\rangle$ with an extra phase shift $\exp\left(2\pi i \frac{az}{2^m}\right)$.

This construction can be readily extended to implement the multiplication unitaries described in Sec. 7.2.1. We simply add the product (cx in the classical-quantum case, xy in the quantum-quantum case) to the $|w\rangle$ register using Fourier addition. Denoting the unitaries that apply the appropriate phase in the Fourier basis with a tilde:

$$\mathcal{U} = (\mathbb{I} \otimes \mathsf{IQFT})\, \tilde{\mathcal{U}}\, (\mathbb{I} \otimes \mathsf{QFT}) \tag{7.21}$$

we can write down $\tilde{\mathcal{U}}_{q\times c}$ and $\tilde{\mathcal{U}}_{q\times q}$ (in terms of their action on product states):

$$\tilde{\mathcal{U}}_{q\times c}(a) |x\rangle |z\rangle = \exp\left(2\pi i \frac{axz}{2^m}\right) |x\rangle |z\rangle \tag{7.22}$$

$$\tilde{\mathcal{U}}_{q\times q} |x\rangle |y\rangle |z\rangle = \exp\left(2\pi i \frac{xyz}{2^m}\right) |x\rangle |y\rangle |z\rangle \tag{7.23}$$

Note that here, z represents each basis state resulting from applying the quantum Fourier transform to the output register; z is not the value itself that was already stored in the output register. We may decompose the quantum values as $x = \sum_i 2^i x_i$ (and respectively

for y and z), yielding:

$$\exp\left(2\pi i\frac{axz}{2^m}\right) = \prod_{i,j}\exp\left(\frac{2\pi i a 2^{i+j}}{2^m}x_i z_j\right) \tag{7.24}$$

$$\exp\left(2\pi i\frac{xyz}{2^m}\right) = \prod_{i,j,k}\exp\left(\frac{2\pi i 2^{i+j+k}}{2^m}x_i y_j z_k\right) \tag{7.25}$$

Since we have decomposed to individual bits, the product $x_i y_j$ (or $x_i y_j z_k$ in the quantum-quantum case) is simply 1 if all relevant bits are 1, or 0 otherwise. This means these individual phase factors can be implemented via (doubly-)controlled-phase rotations of phase

$$\phi_{ij} = \frac{2\pi a 2^{i+j}}{2^m} \pmod{2\pi} \tag{7.26}$$

between qubits x_i and z_j in the classical-quantum case, and

$$\phi_{ijk} = \frac{2\pi 2^{i+j+k}}{2^m} \pmod{2\pi} \tag{7.27}$$

between qubits x_i, y_j, and z_k in the quantum-quantum case. Importantly, these phases depend only on classical values, and can be computed during circuit compilation on a classical computer.

For the classical-quantum multiplication case, the total number of controlled-phase gates applied to implement multiplication is $\mathcal{O}(n^2)$, which matches the complexity achieved by converting the classical "schoolbook" algorithm for integer multiplication into a quantum circuit (but is worse than the sub-quadratic multiplications described earlier). This scaling arises in an interesting way: we have accrued an extra factor of n due to the fact that a rotation proportional to the product ax must be performed for *each* of the n bits of z. However, we have simultaneously managed to get rid of a factor of n, due to the fact that a is classical, so we can coalesce the rotations corresponding to each bit of a and do them as one gate. Unfortunately, for quantum-quantum multiplication, it is not possible to coalesce the rotations for each y_j in the same way, because y is stored in qubits. Thus we need $\mathcal{O}(n^3)$ doubly-controlled-phase gates, which is considerably worse than quantum versions of schoolbook multiplication. Yet there are still benefits to this strategy. First, the constants on the scaling are very good: implementing $\tilde{\mathcal{U}}$ requires *exactly* n^2 or n^3 phase rotations (actually, even fewer since some of the rotations are so small they can be dropped). Second, it uses zero ancilla qubits. This can be contrasted with traditional adder circuits, which need extra work space to perform carries. Finally, it can be massively parallelized, since all the phase rotations commute, yielding better circuit depth.

In this work, we call this type of quantum circuit for multiplication "Fourier multiplication."

7.3 Results

Our main result is that it is possible to decompose the phases used in a Fourier multiplication via Toom-Cook-like transformations, ultimately yielding circuits with fewer than $\mathcal{O}(n^2)$ gates for both classical-quantum and quantum-quantum multiplication. We first give the intuition behind the transformation, explaining it for $k = 2$ for both classical-quantum and quantum-quantum multiplication. Then, we describe the case of arbitrary k in Section 7.3.3. Next, we analyze the gate counts and ancilla usage of these algorithms. Finally we describe how small modifications allow us to optimize the ancilla usage and depth, achieving the results stated in the introduction.

Throughout this work we will focus specifically on the implementation of an operation we call the "PhaseProduct:"

$$\sum_{x,z} c_{x,z} |x\rangle |z\rangle \xRightarrow{\text{PhaseProduct}(\phi)} \sum_{x,z} \exp(i\phi xz) c_{x,z} |x\rangle |z\rangle \tag{7.28}$$

and a related operation which we call the "PhaseTripleProduct:"

$$\sum_{x,y,z} c_{x,y,z} |x\rangle |y\rangle |z\rangle \xRightarrow{\text{PhaseTripleProduct}(\phi)} \sum_{x,y,z} \exp(i\phi xyz) c_{x,y,z} |x\rangle |y\rangle |z\rangle \tag{7.29}$$

Note that these operations are generalizations of $\tilde{\mathcal{U}}_{q\times c}$ and $\tilde{\mathcal{U}}_{q\times q}$ respectively—setting $\phi = 2\pi c/2^m$ in the former and $\phi = 2\pi/2^m$ in the latter recovers Eqs. 7.22 and 7.23. The generalization will be helpful when we apply the transformations recursively.

7.3.1 $k = 2$ classical-quantum multiplication

The fundamental idea is to decompose the phase ϕxz of Eq. 7.28 by using the Karatsuba decomposition (and later via Toom-Cook for general k). A first attempt at directly substituting Eq. 7.7 into the phase of Eq. 7.28 and decomposing into a product of phases yields

$$\exp(2\pi i\phi xz) = \exp(2\pi i\phi 2^n x_1 z_1) \cdot \exp\left(2\pi i\phi 2^{n/2}((x_0+x_1)(z_0+z_1) - x_1z_1 - x_0z_0)\right) \cdot \exp(2\pi i\phi x_0 z_0)$$

This decomposition is not ideal—we still need to compute the value $(x_0 + x_1)(z_0 + z_1) - x_1z_1 - x_0z_0$, which requires storing and reusing the partial products x_0z_0 and x_1z_1. Instead, we note that the prefactors on each phase are just some value, which is classically computed at circuit compilation time. We should endeavor to move as much of the arithmetic into these prefactors as possible, since they are set before the circuit runs and any complexity there does not affect the quantum circuit cost. Thus rearranging Eq. 7.7 to group by partial products, rather than powers of two, yields

$$\exp(2\pi i\phi xz) = \exp\left(2\pi i(2^n - 2^{n/2})\phi x_1 z_1\right) \cdot \exp\left(2\pi i 2^{n/2}\phi(x_0+x_1)(z_0+z_1)\right) \cdot \exp\left(2\pi i(1-2^{n/2})\phi x_0 z_0\right) \tag{7.30}$$

The implications of this expression are drastic. Each phase rotation on the right hand side consists of an entirely classical phase factor, multiplied by a product of two quantum integers. This is simply another application of PhaseProduct, but in which the integers are half as long! We can recursively apply the decomposition again to each of the sub-products, until the input integers are sufficiently small that we perform the phase products directly via individual two-qubit controlled-phase rotations.

Importantly, breaking the phase down this way does not require computing and storing any extra values in ancilla registers—the computation can be performed *in-place*. We do need the values $(x_0 + x_1)$ and $(z_0 + z_1)$, which we compute via regular quantum addition circuits. Importantly however, since addition is reversible we can compute these values in-place (up to a single "overflow" qubit), temporarily overwriting x_1 and z_1 respectively.

In Algorithm 7.1, we explicitly record the steps of this algorithm, and then we move on to the case of quantum-quantum multiplication.

Algorithm 7.1: PhaseProduct(ϕ, $|x\rangle$, $|z\rangle$) for $k = 2$

Input: Quantum state $\sum_{x,z} c_{x,z} |x\rangle |z\rangle$
Classical phase factor ϕ

Output: Quantum state $\sum_{x,z} \exp(2\pi i \phi x z) c_{x,z} |x\rangle |z\rangle$

Divide $|x\rangle$ and $|z\rangle$ registers in half, so $|x\rangle = |x_0\rangle |x_1\rangle$ and $|z\rangle = |z_0\rangle |z_1\rangle$ (here using little-endian bit order).

1 Apply PhaseProduct($(1 - 2^{n/2})\phi, |x_0\rangle, |z_0\rangle$) via recursive call
2 Apply PhaseProduct($(2^n - 2^{n/2})\phi, |x_1\rangle, |z_1\rangle$) via recursive call
3 Apply quantum addition circuit from $|x_0\rangle$ to $|x_1\rangle$, yielding the state $|x_0\rangle |x_0 + x_1\rangle$ in the x register (and resp. for z register)
4 Apply PhaseProduct($2^{n/2}\phi, |x_0 + x_1\rangle, |z_0 + z_1\rangle$) via recursive call
5 Apply quantum subtraction circuit from $|x_0\rangle$ to $|x_0 + x_1\rangle$, restoring it to $|x_1\rangle$ (and resp. for z register)

7.3.2 $k = 2$ quantum-quantum multiplication

For quantum-quantum multiplication we desire to implement the phase shift ϕxyz from Eq. 7.29, where x, y, and z are all stored in quantum registers. In a less constrained setting (such as in classical computing or "digital" quantum arithmetic), the product of three integers would be computed by taking the product of two of the integers and then multiplying the third integer by the result—so no special algorithm for the product of three integers is required. However, we are decomposing a phase, and this is only possible over *sums*, not products. Thus we introduce modified versions of the Karatsuba algorithm, which compute the product of three integers in one step.

Fortunately, viewing Karatsuba as pointwise multiplication of polynomials (as discussed in Sec. 7.2.2) gives a fairly straightforward way to do this. Denote the integer product we desire as $p = xyz$. We will again consider the factors x, y, and z as polynomials and evaluate them at a set of points w_i. This time we will compute the pointwise *triple* product $p(w_i) = x(w_i)y(w_i)z(w_i)$, and as before interpolate the polynomial $p(w)$ and evaluate it at the appropriate w to recover the desired integer result p. The main difference from the two-integer case is that the degree of the polynomial p will be higher (3 in this case), so we will need to use more points w_i. Regular (two-integer) Karatsuba as described in Sec. 7.2.2 uses the points $\{0, \infty, 1\}$, fortunately, it is easy to include the point -1 corresponding to the linear combination $x_0 - x_1$ (resp. y and z), for a total of $3 + 1 = 4$ points which uniquely will specify our degree-3 polynomial p. With that, we can write down a modified Karatsuba decomposition which applies to triple products (here grouping by partial products in view of the discussion in classical-quantum case):

$$xyz = (2^{3n/2} - 2^{n/2})x_1y_1z_1 + \frac{1}{2}(2^n + 2^{n/2})(x_0 + x_1)(y_0 + y_1)(z_0 + z_1) + \frac{1}{2}(2^n - 2^{n/2})(x_0 - x_1)(y_0 - y_1)(z_0 - z_1) + (1 - 2^n)x_0y_0z_0 \quad (7.31)$$

As before, the factors in each of the sub-multiplications are half the size of the original ones (up to one bit of overflow)!

Using this to decompose the phase ϕxyz, we end up with an algorithm that is structurally identical to the recursive decomposition of ϕxz in the classical-quantum case. In Algorithm 7.2, we describe how to perform this decomposition explicitly.

Note that as we show in Table 7.1, the $k = 2$ quantum-quantum case does not actually provide a sub-quadratic gate count—it requires $\mathcal{O}(n^2)$ two-qubit gates, matching the asymptotic cost of "digital" schoolbook multiplication. However it does provide a drastic speedup over the $\mathcal{O}(n^3)$ gate count of the schoolbook algorithm for the PhaseTripleProduct operation represented in Eq. 7.29. In any case, it's an illustrative example that provides good intuition for the $k > 2$ cases which do yield a sub-quadratic runtime, which we discuss next.

7.3.3 $k > 2$

In this section we describe the general case of $k > 2$ for both classical-quantum and quantum-quantum multiplication. As in the previous two sections, we would like to apply the Toom-Cook decomposition to the phase products ϕxz and ϕxyz. Using the notation introduced in Sec. 7.2.2.2, we can write the entire Toom-Cook process for computing an integer product xz as a linear algebra expression:

$$xz = (\boldsymbol{e}(2^{n/k}))^\intercal A^{-1}(A\boldsymbol{x} \circ A\boldsymbol{z}) \quad (7.32)$$

As in the previous section, to decompose the triple product we modify the Toom-Cook algorithm to take the pointwise triple product:

$$xyz = (\boldsymbol{e}(2^{n/k}))^\intercal A^{-1}(A\boldsymbol{x} \circ A\boldsymbol{y} \circ A\boldsymbol{z}) \quad (7.33)$$

Algorithm 7.2: PhaseTripleProduct(ϕ, $|x\rangle$, $|y\rangle$, $|z\rangle$) for $k = 2$

Input: Quantum state $\sum_{x,y,z} c_{x,y,z} |x\rangle |y\rangle |z\rangle$
Classical phase factor ϕ
Output: Quantum state $\sum_{x,y,z} \exp(i\phi xyz) c_{x,y,z} |x\rangle |y\rangle |z\rangle$

Divide $|x\rangle$, $|y\rangle$, and $|z\rangle$ registers in half, so $|x\rangle = |x_0\rangle |x_1\rangle$, $|y\rangle = |y_0\rangle |y_1\rangle$ and $|z\rangle = |z_0\rangle |z_1\rangle$ (here using little-endian bit order).

1 Apply PhaseTripleProduct($(1 - 2^n)\phi, |x_0\rangle, |y_0\rangle, |z_0\rangle$) via recursive call
2 Apply PhaseTripleProduct($(2^{3n/2} - 2^{n/2})\phi, |x_1\rangle, |y_1\rangle, |z_1\rangle$) via recursive call
3 Apply quantum addition circuit from $|x_1\rangle$ to $|x_0\rangle$, yielding the state $|x_0 + x_1\rangle |x_1\rangle$ in the x register (and resp. for y and z registers)
4 Apply PhaseTripleProduct($\frac{1}{2}(2^n + 2^{n/2})\phi, |x_0 + x_1\rangle, |y_0 + y_1\rangle, |z_0 + z_1\rangle$) via recursive call
5 Apply quantum circuit to subtract $2x_1$ from $|x_0 + x_1\rangle$, yielding the state $|x_0 - x_1\rangle |x_1\rangle$ in the x register (and resp. for y and z registers)
6 Apply PhaseTripleProduct($\frac{1}{2}(2^n - 2^{n/2})\phi, |x_0 - x_1\rangle, |y_0 - y_1\rangle, |z_0 - z_1\rangle$) via recursive call
7 Apply quantum addition circuit from $|x_1\rangle$ to $|x_0 - x_1\rangle$, restoring it to $|x_0\rangle$ (and resp. for y and z registers)

Note that as in Section 7.3.2, the triple product increases the degree of the product polynomial, so we must evaluate the polynomials at more points. As a function of k, this number of points is $q = 3(k-1)+1$ in the three-integer (PhaseTripleProduct) case (whereas it is $q = 2(k-1)+1$ in the two-integer (PhaseProduct) case)

Inserting these expressions into the phases of Eq. 7.28 and 7.29, we have

$$\exp(i\phi x[y]z) = \exp\left(i\phi(\boldsymbol{e}(2^{n/k}))^\intercal A^{-1}(A\boldsymbol{x}[\circ A\boldsymbol{y}] \circ A\boldsymbol{z})\right) \tag{7.34}$$

where we use square brackets to denote parts that only appear in the three-integer, PhaseTripleProduct case, but not in the two integer PhaseProduct case.

As in the $k = 2$ case, we'd like to decompose Eq. 7.34 in a way that allows us to include as much linear algebra as possible in the prefactors, which can be classically computed at circuit compilation time. To do so, we define $\hat{\boldsymbol{e}}^\intercal = (\boldsymbol{e}(2^{n/k}))^\intercal A^{-1}$, and decompose the phase across the inner product of $\hat{\boldsymbol{e}}^\intercal$ and the pointwise product $A\boldsymbol{x}[\circ A\boldsymbol{y}] \circ A\boldsymbol{z}$:

$$\exp(i\phi x[y]z) = \prod_i \exp(i\phi \hat{e}_i (A\boldsymbol{x})_i [(A\boldsymbol{y})_i](A\boldsymbol{z})_i) \tag{7.35}$$

We see that now, as before, each of the phase factors in the product on the right hand side has the same form as the left hand side—a classical phase factor (in this case $\phi\hat{w}_i$) multiplied by a product of integers. By choosing the points w_i appropriately we can ensure that the

values $(A\boldsymbol{x})_i$ (resp. y and z) are roughly $1/k$ the size of the input integers, up to overflow bits, reducing our larger multiplication to q much smaller ones.

The decomposition in Eq. 7.35 suggests a simple algorithm for computing PhaseProduct and PhaseTripleProduct efficiently, for general k. We lay out the steps explicitly in Algorithm 7.3. In that algorithm, the square brackets denote values that only appear when computing the PhaseTripleProduct. We note that Algorithm 7.3 is meant to be as simple as possible while exhibiting the asymptotic scaling we desire; in practice optimizations should be applied to reduce the qubit count and/or depth (see Secs. 7.3.4.2 and 7.3.4.3 below).

Algorithm 7.3: Phase[Triple]Product(ϕ, $|x\rangle$, [$|y\rangle$], $|z\rangle$)

Input: Quantum state $\sum_{x,[y,]z} c_{x,[y,]z} |x\rangle [|y\rangle] |z\rangle$
Classical phase factor ϕ
Tunable parameter k (and corresponding data A, $\boldsymbol{w}$, etc.)
Output: Quantum state $\sum_{x,[y,]z} \exp(i\phi x[y]z) c_{x,[y,]z} |x\rangle [|y\rangle] |z\rangle$

Divide $|x\rangle$, [$|y\rangle$,] and $|z\rangle$ registers into k subregisters each.
Let $q = 2(k-1)+1$ [$q = 3(k-1)+1$]

for i *in* $1:q$ **do**
  Let h be an index such that $A_{ih} = 1$ `/* such an index always exists */`
1 Use quantum addition/subtraction to overwrite x_h (and a few "overflow" ancillas) with $(A\boldsymbol{x})_i$ (resp. [y,] z)
2 Apply Phase[Triple]Product($\hat{w}_i\phi, (A\boldsymbol{x})_i, [(A\boldsymbol{y})_i,](A\boldsymbol{z})_i$) via recursive call
3 Use quantum addition/subtraction to return $(A\boldsymbol{x})_i$ to x_h (resp. [y,] z)
end

Finally, note that the multiplications by $\boldsymbol{e}(2^{n/k})^\intercal$ and A^{-1} in Eqs. 7.32 and 7.33 correspond to step 4 (interpolation) and step 5 (recomposition) of the classical Toom-Cook algorithm respectively (see Sec. 7.2.2). Step 4 is the most complicated of the whole algorithm in terms of arithmetic; it requires roughly twice as many additions and subtractions as step 2 in the two-integer product case, and three times as many in the triple product. By offloading that step to classical precomputation, we have cut the quantum computational cost of a each level of recursion by an additional roughly 2/3 (PhaseProduct) or 3/4 (PhaseTripleProduct).

7.3.4 Analysis of asymptotic costs

7.3.4.1 Gate counts

We now analyze the asymptotic scaling of the gate counts for Algorithm 7.3. We state our results as the following Theorem, which yields the scaling exponents listed in Table 7.1. Its proof is similar to the proof of the asymptotic complexity of the classical Toom-Cook algorithms. [225]

Algorithm	Gate count
Schoolbook	$\mathcal{O}(n^2)$
$k=2$	$\mathcal{O}(n^{\log_2 3}) = \mathcal{O}(n^{1.58\cdots})$
$k=3$	$\mathcal{O}(n^{\log_3 5}) = \mathcal{O}(n^{1.46\cdots})$

(a) Classical-quantum multiplication

Algorithm	Gate count
Schoolbook	$\mathcal{O}(n^3)$
$k=2$	$\mathcal{O}(n^{\log_2 4}) = \mathcal{O}(n^2)$
$k=3$	$\mathcal{O}(n^{\log_3 7}) = \mathcal{O}(n^{1.77\cdots})$
$k=4$	$\mathcal{O}(n^{\log_4 10}) = \mathcal{O}(n^{1.66\cdots})$

(b) Quantum-quantum multiplication

Table 7.1: **Asymptotic scaling of gate counts for Algorithm 7.3, for various k relevant in practice.** "Schoolbook" in this table refers to the phase decompositions of Eqs. 7.24 and 7.25.

Theorem 6. *Algorithm 7.3 can be implemented on a quantum device using $\mathcal{O}(n^{\log_k q})$ two-qubit gates, where n is the length of the input integers.*

Proof. We first show that the recursive calls are performed on integers of size $n/k + t(k)$, for some function $t(k)$ which is independent of n. From this we write a recursion relation and solve it to prove the theorem.

The recursive calls are performed on values of the form $(A\boldsymbol{x})_i$ (resp. y and z). For the purposes of this proof, let the points w_i be any q integers in the range $(-q, q)$. Then, the relevant elements of A—corresponding to powers of each point w_i up to degree k—are bounded by q^k; since q is linear in k this implies that the elements of A are at most $\mathcal{O}(k \log k)$ bits long. The elements of $\boldsymbol{x}$ (resp. y and z) are of size n/k, thus a single product $A_{ij}x_j$ has length $n/k + \mathcal{O}(k \log k)$. The value $(A\boldsymbol{x})_i$ is the sum over k of these products, which adds at most $\log k$ bits, so we see that $(A\boldsymbol{x})_i$ can be stored in $n/k + \mathcal{O}(k \log k)$ bits. Thus we see $t(k) = \max_i \lceil \log(A\boldsymbol{x})_i \rceil - n/k = \mathcal{O}(k \log k)$.

With that we may write a recursion relation for the gate count G. For one level of recursion, the algorithm performs at most $2qk$ additions and subtractions, and q recursive calls of size $n/k + t(k)$ (one per iteration of the outer loop). With the additions and subtractions being performed in linear time dn for some constant d, we have

$$G(n) = dn + qG(n/k + t(k)) \tag{7.36}$$

Noting that if G is a polynomial of degree ≤ 2, then $G(n/k + t(k)) \leq G(n/k) + \mathcal{O}(nt(k))$, solving the recursion yields

$$G(n) = \mathcal{O}(n^{\log_k q}) \tag{7.37}$$

which is what we desired to show. Finally, because q is at most $3(k-1)+1$ it is straightforward to see that for all $k \geq 2$ (which is all of the allowed values of k), the power of n is ≤ 2, and thus our assumption that G was a polynomial of degree at most two was justified. □

7.3.4.2 Qubit counts

In the proof of Theorem 6, we showed that the value $(A\boldsymbol{x})_i$ (resp. y and z) may overflow the register it is overwriting by a number of bits $t(k) = \mathcal{O}(k \log k)$ which is constant in n. It is straightforward to see, then, that the algorithm can be executed using $\mathcal{O}(\log n)$ ancilla qubits—a constant number of ancillas are used at each of the $\log n$ levels of recursion. In this section we show that it is possible to do even better, using only a small constant number of qubits for the entire operation (2 for classical-quantum multiplication, and 5 for quantum-quantum).

The first key observation is that we may reduce the overflow to a single qubit, by adjusting slightly how we divide the factors into their k parts. If we set the size of the $k-1$ less-significant parts of the value to $\lfloor (n-t)/k \rfloor$ bits, then the most-significant part will be at least t bits longer than all the others. Thus if we choose our w_i such that the coefficient on that large piece is 1, and overwrite it with the linear combination $(A\boldsymbol{x})_i$ (resp. y and z), it will overflow by at most one qubit. Since each of the x, y, and z will have such an overflow qubit, this leads to a total of 3 overflow qubits per level of recursion (or 2 for the classical-quantum case, in which we only have x and z).

While we now only need one overflow qubit per level of recursion, we still need to deal with the larger issue of overflow qubits accumulating through the recursive calls. To do so, we perform the portion of the multiplication that involves that extra overflow qubit directly, before moving down in the recursive tree. Because one of the factors in this partial multiplication is only a single qubit, the contribution to the overall runtime is negligible. Once it has been done, the overflow qubit is uncomputed, and can be reused at a lower level recursive call.

We make these two ideas explicit in Algorithm 7.4, which reduces the ancilla qubit count to at most 3, while maintaining the asymptotic gate count of Algorithm 7.3.

Theorem 7. *Algorithm 7.4 can be implemented in $\mathcal{O}(n^{\log_k q})$ two-qubit gates, where n is the length of the input integers.*

Proof. There are three important ways in which Algorithm 7.4 differs from Algorithm 7.3: there is only one overflow qubit per register, steps 2-5 are new, and the recursive call of step 6 is on a value of length ℓ' rather than $n/k + t(k)$. For the last one, it is straightforward to see that $\ell' \leq n/k + t(k)$, so the latter change will not adversely affect the running time. Thus we just need to show that only one overflow qubit is required, and that the cost of steps 2-5 is negligible in the asymptotic limit.

First we show that only a single overflow qubit is indeed required to store $(A\boldsymbol{x})_i$ in the subregister x_{k-1} (resp. y and z). First observe that for $w_i \in \{0, \infty\}$, $(A\boldsymbol{x})_i \in x_0, x_{k-1}$ so clearly no overflow qubits are needed. The other w_i are unit fractions with denominator α_i. For these, recall that $(A\boldsymbol{x})_i = \sum_j \alpha_i^{k-1-j} x_j$. Thus $(A\boldsymbol{x})_i$ is bounded from above by $2^{\ell'} + 2^{\ell} t'$. By definition $\ell' - \ell \geq t'$, so $(A\boldsymbol{x})_i$ requires at most $\ell' + 1$ bits to store, which is what we desired to show.

Algorithm 7.4: Phase[Triple]Product(ϕ, $|x\rangle$, [$|y\rangle$], $|z\rangle$) with $\mathcal{O}(1)$ ancillas

Input: Quantum state $\sum_{x,[y,]z} c_{x,[y,]z} |x\rangle [|y\rangle] |z\rangle$
Classical phase factor ϕ
Tunable parameter k

Output: Quantum state $\sum_{x,[y,]z} \exp(i\phi x[y]z) c_{x,[y,]z} |x\rangle [|y\rangle] |z\rangle$

Let $\alpha_{\max} = \lceil (k-2)/2 \rceil$.
Let the evaluation points w_i be k values from the set
$\{0, \infty\} \cup \{1/\alpha, -1/\alpha | 1 \leq \alpha \leq \alpha_{\max}\}$.
Let $t' = \lceil \log(\sum_{1 \leq i < k} {\alpha_{\max}}^i) \rceil$.
Let $\ell = \lfloor (n - t')/k \rfloor$.
Let $q = 2(k-1) + 1$ [$q = 3(k-1) + 1$].
Divide $|x\rangle$, [$|y\rangle$,] and $|z\rangle$ registers into k subregisters, where the $k-1$ least significant have length ℓ and the most significant contains the remaining bits whose size we denote ℓ'.

for *i in* $1 : q$ **do**

1 Use quantum addition/subtraction to overwrite x_{k-1} and one "overflow" ancilla with $(A\boldsymbol{x})_i$ (resp. [y,] z)
Let $|x_\omega\rangle$ represent the overflow qubit of the x_{k-1} subregister (resp. y, z)
2 Apply phase $2^{\ell'} \hat{w}_i \phi [(A\boldsymbol{y})_i](A\boldsymbol{z})_i$ controlled by qubit $|x_\omega\rangle$
3 [Apply phase $2^{\ell'} \hat{w}_i \phi ((A\boldsymbol{x})_i \bmod 2^{\ell'})(A\boldsymbol{z})_i$ controlled by qubit $|y_\omega\rangle$]
4 Apply phase $2^{\ell'} \hat{w}_i \phi ((A\boldsymbol{x})_i \bmod 2^{\ell'})[((A\boldsymbol{y})_i \bmod 2^{\ell'})]$ controlled by qubit $|z_\omega\rangle$
5 Uncompute the overflow qubit for each register
6 Apply Phase[Triple]Product$(\hat{w}_i \phi, (A\boldsymbol{x})_i \bmod 2^{\ell'}, [(A\boldsymbol{y})_i \bmod 2^{\ell'},](A\boldsymbol{z})_i \bmod 2^{\ell'})$ via recursive call
7 Use quantum addition/subtraction to return $(A\boldsymbol{x})_i \bmod 2^{\ell'}$ to x_h (resp. [y,] z)

end

Now we examine the cost of steps 2-5. We begin with steps 2-4, in the case of PhaseProduct (used for classical-quantum multiplication), because it is simpler. In this case the parts of Algorithm 7.4 in square brackets are omitted, including all of step 3. In steps 2 and 4, we need to do a controlled application of phases proportional to $(A\boldsymbol{z})_i$ and $(A\boldsymbol{x})_i \bmod 2^{\ell'}$. These values are both already contained in a register, so this can be easily achieved with a series of ℓ' controlled-phase gates, where ℓ' is within in additive constant of n/k. Thus in this case our recursion relation is simply (cf. Eq 7.36)

$$G(n) = \mathcal{O}(n) + qG(\ell') \tag{7.38}$$

For steps 2-4 of PhaseTripleProduct, we need to implement phases proportional to $(A\boldsymbol{y})_i (A\boldsymbol{z})_i$, $((A\boldsymbol{x})_i \bmod 2^{n/k})(A\boldsymbol{z})_i$, and $((A\boldsymbol{x})_i \bmod 2^{n/k})((A\boldsymbol{y})_i \bmod 2^{n/k})$. Once again all these values are readily available, but simply applying a series of doubly-controlled phase

rotations would use n^2 gates—which is too large. Instead, we may apply a controlled version of the PhaseProduct algorithm, for some (perhaps different) k'! Doing so will require a total of $\mathcal{O}(n^{\log'_k q'})$ gates, and we get a recursion relation

$$G(n) = \mathcal{O}(n^{\log'_k q'}) + qG(\ell') \tag{7.39}$$

where again ℓ' is within an additive constant of n/k. Thus as long as we set k' such that

$$\log_{k'}[2(k'-1)+1] < \log_k[3(k-1)+1] \tag{7.40}$$

both Eqs. 7.38 and 7.39 have the solution

$$\mathcal{O}(n^{\log_k q}) \tag{7.41}$$

which is what we desired to show.

The only thing that remains is step 5. We must show that uncomputing the overflow ancillas uses only a negligible number of gates. This is straightforward: the *entire* value $(A\boldsymbol{x})_i$ requires only a number of gates linear in n to compute, so uncomputing the entire thing and recomputing $(A\boldsymbol{x})_i \bmod 2^{\ell'}$ would require only $\mathcal{O}(n)$ gates. (In practice the single ancilla can be uncomputed more efficiently than uncomputing the entire value). □

Remark: We note that $k' = 2$ satisfies Eq. 7.40 for $k \leq 5$, which includes all of the k values we explicitly explore in this work.

Theorem 8. *Algorithm 7.4 uses at most 2 ancillas for PhaseProduct, and at most 5 for PhaseTripleProduct.*

Proof. Again we begin with the case of PhaseProduct (which implements classical-quantum multiplication). In step 1, one ancilla qubit is allocated for the x and z registers respectively, for a total of two. Steps 2-4 do not require the allocation of any new ancilla qubits if they are implemented as a series of controlled-phase gates (as they are in the proof of Theorem 7). Because we uncompute the overflow ancillas in step 5, they can then be reused during the recursive call of step 6, thus requiring no additional ancilla qubits to be allocated. Thus PhaseProduct requires a total of 2 ancillas.

For PhaseTripleProduct, 3 ancillas (one for each register) are allocated in step 1. But as described in the proof of Theorem 7, steps 2-4 require the application of a controlled version of PhaseProduct. We have just shown that this requires 2 ancillas. Once again, in step 5 we uncompute the overflow ancillas and they can be reused during the recursive call of Step 6. Thus we see that PhaseTripleProduct can be implemented with a total of 5 ancillas. □

In practice, we may be interested in the size of t', which is roughly $\ell' - n/k$, because although it is clear that their difference is constant, if that constant is large it could meaningfully affect the practical cost of performing the recursion (since the recursive products are of size ℓ'). In Table 7.2, we record the values of t' for various k, and observe that they are very small in practice.

Algorithm variant	t'
$k = 2$	0
$k = 3$	3

(a) Classical-quantum multiplication

Algorithm variant	t'
$k = 2$	0
$k = 3$	4
$k = 4$	7

(b) Quantum-quantum multiplication

Table 7.2: **Number of bits t' by which the length of subregister x_{k-1} must exceed the lengths of the other subregisters in Algorithm 7.4.** This corresponds roughly to the "extra" length in addition to n/k of values which are recursively multiplied.

7.3.4.3 Gate depth

Finally, we discuss gate depth. The main results of this section are summarized in Table 7.3, and formalized in Theorem 9. Stated succinctly, we find that it is possible to perform the phase shifts implemented by PhaseProduct and PhaseTriple product in sub-linear gate depth. Unfortunately, this does not immediately imply that these algorithms can be used to achieve sub-linear depth multiplication in $\mathcal{O}(n)$ qubits—surprisingly, we are bottlenecked by the quantum Fourier transform and its inverse, that need to be performed before and after the phase products! The lowest-depth (approximate) quantum Fourier transform circuit of which we are aware that runs in linear space is the "standard" construction having depth $\mathcal{O}(n)$; fast algorithms which have $\mathcal{O}(\log n)$ depth are known but seem to require at least $\mathcal{O}(n \log n)$ qubits [233]. An important and exciting direction for future research is whether it is possible to compute this (approximate) quantum Fourier transform in sub-linear depth and linear space. Despite this limitation, we believe it is worth exploring how the Phase[Triple]Product portion of the multiplication can be made as low-depth as possible; in practice the constants are good enough on the linear-space quantum Fourier transform circuit that we expect its depth to be comparable to the depth of the Phase[Triple]Product for relevant sizes of integers, despite its worse asymptotic scaling. The low-depth constructions given here use $\mathcal{O}(n)$ ancilla qubits (note that this is more than the $\mathcal{O}(1)$ ancillas we used in the previous section) We expect that in practice the qubit cost will be dominated by the $\mathcal{O}(n)$ qubits needed to perform sub-linear time addition [230]; the number of ancillas needed for the overflow qubits in this case is also linear in n but small. We begin by giving intuition for how low depth can be achieved, and then state Theorem 9.

In Algorithms 7.3 and 7.4, in each iteration of the outer loop overwrites a single value x_h of length n/k with $(A\boldsymbol{x})_i$ (resp. y and z), recursively computes Phase[Triple]Product on that value, and then resets it to its original contents. However, we have k of these length n/k subregisters at our disposal—we may as well overwrite more than just one of them, so that we can perform several recursive calls in parallel! The challenge is that we need to compute each of the $(A\boldsymbol{x})_i$ (resp. y and z) in-place, reversibly. We do not know of an algorithmic way to simultaneously compute many $(A\boldsymbol{x})_i$ in-place for arbitrary k; instead, in Tables 7.4, 7.5, and 7.6 we record explicit sequences of operations to do so, which were

Algorithm	Sequence	Depth
Schoolbook		$\mathcal{O}(n)$
$k = 2$	Alg. 7.1	$\mathcal{O}(n^{\log_2 2}) = \mathcal{O}(n)$
$k = 3$	Table 7.4	$\mathcal{O}(n^{\log_3 2}) = \mathcal{O}(n^{0.63\cdots})$

(a) Classical-quantum multiplication

Algorithm	Sequence	Depth
Schoolbook		$\mathcal{O}(n^2)$
$k = 2$	Alg. 7.2	$\mathcal{O}(n^{\log_2 2}) = \mathcal{O}(n)$
$k = 3$	Table 7.5	$\mathcal{O}(n^{\log_3 3}) = \mathcal{O}(n)$
$k = 4$	Table 7.6	$\mathcal{O}(n^{\log_4 4}) = \mathcal{O}(n)^*$

(b) Quantum-quantum multiplication

Table 7.3: **Asymptotic scaling of gate depth.** "Schoolbook" in this table refers to the phase decompositions of Eqs. 7.24 and 7.25. The form of the scaling relation comes from Theorem 9. All algorithms listed here use only $\mathcal{O}(n)$ qubits. * Sublinear depth should be achievable for $k = 4$ by finding a sequence of operations using fewer parallel groups than that of Table 7.6.

created by hand for the cases of $k = 3$ (both PhaseProduct and PhaseTripleProduct) and $k = 4$ (PhaseTripleProduct only). We also note that Algorithms 7.1 and 7.2, which cover the $k = 2$ cases, already implicitly allow this parallelism. In all of these sequences we have endeavored to only use addition, sign inversion, and scaling by powers of two, since each of these operations should be very efficient to implement.

Since we have k subregisters at our disposal, and q points at which we need to evaluate $(A\boldsymbol{x})_i$ (resp. y and z), the best parallelism we can hope to achieve (in a single level of recursion) is $n_p = \lceil q/k \rceil$ parallel groups of recursive calls. Recalling the dependence of q on k, this quantity is equal to $n_p = \left\lceil \frac{2(k-1)+1}{k} \right\rceil = 2$ for PhaseProduct (for all $k \geq 2$) and $n_p = \left\lceil \frac{3(k-1)+1}{k} \right\rceil = 3$ for PhaseTripleProduct (for all $k \geq 3$). (Note that the optimum is achieved by the sequences in Algs. 7.1 and 7.2, and Tables 7.4 and 7.5). In view of Theorem 9 below, this corresponds to an asymptotic gate depth of $\mathcal{O}(n^{\log_k 2})$ for PhaseProduct and $\mathcal{O}(n^{\log_k 3})$ for PhaseTripleProduct. An avenue for further research is whether it is possible to saturate the hard lower bound of depth equal to total gate count over n, using only $\mathcal{O}(n)$ ancillas, by parallelizing over multiple levels of recursion simultaneously.

We formalize our results regarding gate depth in the following theorem, and give explicit asymptotic scaling of gate depth in Table 7.3.

Operation	Register 0	Register 1	Register 2
(start)	$\|x_0\rangle$	$\|x_1\rangle$	$\|x_2\rangle$
Add reg. 2 to reg. 1	$\|x_0\rangle$	$\|x_2 + x_1\rangle$	$\|x_2\rangle$
Add reg. 0 to reg. 1	$\|x_0\rangle$	$\|x_2 + x_1 + x_0\rangle$	$\|x_2\rangle$
Product on all registers	$\boldsymbol{\|x_0\rangle}$	$\boldsymbol{\|x_2 + x_1 + x_0\rangle}$	$\boldsymbol{\|x_2\rangle}$
Invert sign of reg. 1	$\|x_0\rangle$	$\|-x_2 - x_1 - x_0\rangle$	$\|x_2\rangle$
Add reg. 0 to reg. 1	$\|x_0\rangle$	$\|-x_2 - x_1\rangle$	$\|x_2\rangle$
Add 2× reg. 2 to reg. 1	$\|x_0\rangle$	$\|x_2 - x_1\rangle$	$\|x_2\rangle$
Add reg. 1 to reg. 0	$\|x_2 - x_1 + x_0\rangle$	$\|x_2 - x_1\rangle$	$\|x_2\rangle$
Add reg. 0 to reg. 1	$\|x_2 - x_1 + x_0\rangle$	$\|2x_2 - 2x_1 + x_0\rangle$	$\|x_2\rangle$
Add 2× reg. 2 to reg. 1	$\|x_2 - x_1 + x_0\rangle$	$\|4x_2 - 2x_1 + x_0\rangle$	$\|x_2\rangle$
Product on regs. 1 and 0	$\boldsymbol{\|x_2 - x_1 + x_0\rangle}$	$\boldsymbol{\|4x_2 - 2x_1 + x_0\rangle}$	$\|x_2\rangle$
Invert sign of reg. 1	$\|x_2 - x_1 + x_0\rangle$	$\|-4x_2 + 2x_1 - x_0\rangle$	$\|x_2\rangle$
Add 2× reg. 2 to reg. 1	$\|x_2 - x_1 + x_0\rangle$	$\|-2x_2 + 2x_1 - x_0\rangle$	$\|x_2\rangle$
Add reg. 1 to 2× reg. 0	$\|x_0\rangle$	$\|-2x_2 + 2x_1 - x_0\rangle$	$\|x_2\rangle$
Add reg. 0 to reg. 1	$\|x_0\rangle$	$\|-2x_2 + 2x_1\rangle$	$\|x_2\rangle$
Add 2× reg. 2 to reg. 1	$\|x_0\rangle$	$\|2x_1\rangle$	$\|x_2\rangle$
Divide reg. 1 by two	$\|x_0\rangle$	$\|x_1\rangle$	$\|x_2\rangle$

Table 7.4: **$k = 3$ low-depth classical-quantum multiplication algorithm.** This table lists the quantum operations performed to implement the unitary $\tilde{\mathcal{U}}_{q\times c}(a)$. The registers are divided into subregisters as $|x\rangle = |x_0\rangle\,|x_1\rangle\,|x_2\rangle$ and $|z\rangle = |z_0\rangle\,|z_1\rangle\,|z_2\rangle$ (using little-endian notation, so x_0 is the least-significant subregister). In this table only the state of the x subregisters are shown; the same operations are applied to the z register. "Product on registers" means to apply a phase corresponding to the product of the respective x and z registers, usually by recursively calling the same algorithm again. Registers containing values upon which the algorithm is applied recursively are highlighted in bold. The linear combinations used here for the products correspond to the evaluation points $w_i \in \{0, \infty, \pm 1, -2\}$.

Operation	Register 0	Register 1	Register 2
(start)	$\|x_0\rangle$	$\|x_1\rangle$	$\|x_2\rangle$
Add reg. 0 to reg. 1	$\|x_0\rangle$	$\|x_0 + x_1\rangle$	$\|x_2\rangle$
Add reg. 2 to reg. 1	$\|x_0\rangle$	$\|x_0 + x_1 + x_2\rangle$	$\|x_2\rangle$
Product on all	$\|\boldsymbol{x_0}\rangle$	$\|\boldsymbol{x_0 + x_1 + x_2}\rangle$	$\|\boldsymbol{x_2}\rangle$
Add $-1\times$ reg. 2 to reg. 1	$\|x_0\rangle$	$\|x_0 + x_1\rangle$	$\|x_2\rangle$
Add $2\times$ reg. 0 to $-1\times$ reg. 1	$\|x_0\rangle$	$\|x_0 - x_1\rangle$	$\|x_2\rangle$
Add reg. 1 to reg. 2	$\|x_0\rangle$	$\|x_0 - x_1\rangle$	$\|x_0 - x_1 + x_2\rangle$
Add reg. 2 to reg. 1	$\|x_0\rangle$	$\|2x_0 - 2x_1 + x_2\rangle$	$\|x_0 - x_1 + x_2\rangle$
Add $2\times$ reg. 0 to reg. 1	$\|x_0\rangle$	$\|4x_0 - 2x_1 + x_2\rangle$	$\|x_0 - x_1 + x_2\rangle$
Product on regs. 1 and 2	$\|x_0\rangle$	$\|\boldsymbol{4x_0 - 2x_1 + x_2}\rangle$	$\|\boldsymbol{x_0 - x_1 + x_2}\rangle$
Add reg. 0 to $2\times$ reg. 2	$\|x_0\rangle$	$\|4x_0 - 2x_1 + x_2\rangle$	$\|3x_0 - 2x_1 + 2x_2\rangle$
Add reg. 2 to $-2\times$ reg. 1	$\|x_0\rangle$	$\|-5x_0 + 2x_1\rangle$	$\|3x_0 - 2x_1 + 2x_2\rangle$
Add $2\times$ reg. 2 to reg. 1	$\|x_0\rangle$	$\|x_0 - 2x_1 + 4x_2\rangle$	$\|3x_0 - 2x_1 + 2x_2\rangle$
Add reg. 1 to $-4\times$ reg. 2	$\|x_0\rangle$	$\|x_0 - 2x_1 + 4x_2\rangle$	$\|-11x_0 + 6x_1 - 4x_2\rangle$
Add $2\times$ reg. 1 to reg. 2	$\|x_0\rangle$	$\|x_0 - 2x_1 + 4x_2\rangle$	$\|-9x_0 + 2x_1 + 4x_2\rangle$
Add $8\times$ reg. 0 to reg. 2	$\|x_0\rangle$	$\|x_0 - 2x_1 + 4x_2\rangle$	$\|-x_0 + 2x_1 + 4x_2\rangle$
Add $2\times$ reg. 0 to reg. 2	$\|x_0\rangle$	$\|x_0 - 2x_1 + 4x_2\rangle$	$\|x_0 + 2x_1 + 4x_2\rangle$
Product on regs. 1 and 2	$\|x_0\rangle$	$\|\boldsymbol{x_0 - 2x_1 + 4x_2}\rangle$	$\|\boldsymbol{x_0 + 2x_1 + 4x_2}\rangle$
Add reg. 2 to $-1\times$ reg. 1	$\|x_0\rangle$	$\|4x_1\rangle$	$\|x_0 + 2x_1 + 4x_2\rangle$
Add $-1\times$ reg. 0 to reg. 2	$\|x_0\rangle$	$\|4x_1\rangle$	$\|2x_1 + 4x_2\rangle$
Add $-1\times$ reg. 1 to $2\times$ reg. 2	$\|x_0\rangle$	$\|4x_1\rangle$	$\|8x_2\rangle$
Divide reg. 1 by 4	$\|x_0\rangle$	$\|x_1\rangle$	$\|8x_2\rangle$
Divide reg. 2 by 8	$\|x_0\rangle$	$\|x_1\rangle$	$\|x_2\rangle$

Table 7.5: **$k = 3$ quantum-quantum multiplication sequence.** In this table only the state of the x sub-registers are shown; the same operations are applied to the z register. Registers containing values upon which the algorithm is applied again recursively are highlighted in bold. The linear combinations used here for the products correspond to the evaluation points $w_i \in \{0, \infty, \pm 1, \pm 2, -1/2\}$.

Operation	Register 0	Register 1	Register 2	Register 3
(start)	$\|x_0\rangle$	$\|x_1\rangle$	$\|x_2\rangle$	$\|x_3\rangle$
Add reg. 0 to reg. 2	$\|x_0\rangle$	$\|x_1\rangle$	$\|x_0 + x_2\rangle$	$\|x_3\rangle$
Add reg. 3 to reg. 1	$\|x_0\rangle$	$\|x_1 + x_3\rangle$	$\|x_0 + x_2\rangle$	$\|x_3\rangle$
Add reg. 2 to reg. 1	$\|x_0\rangle$	$\|x_0 + x_1 + x_2 + x_3\rangle$	$\|x_0 + x_2\rangle$	$\|x_3\rangle$
Add reg. 1 to $-2\times$ reg. 2	$\|x_0\rangle$	$\|x_0 + x_1 + x_2 + x_3\rangle$	$\|-x_0 + x_1 - x_2 + x_3\rangle$	$\|x_3\rangle$
Product on all regs.	$\|\boldsymbol{x_0}\rangle$	$\|\boldsymbol{x_0 + x_1 + x_2 + x_3}\rangle$	$\|\boldsymbol{-x_0 + x_1 - x_2 + x_3}\rangle$	$\|\boldsymbol{x_3}\rangle$
Add reg. 1 to reg. 2	$\|x_0\rangle$	$\|x_0 + x_1 + x_2 + x_3\rangle$	$\|2x_1 + 2x_3\rangle$	$\|x_3\rangle$
Add $-3\times$ reg. 3 to reg. 2	$\|x_0\rangle$	$\|x_0 + x_1 + x_2 + x_3\rangle$	$\|2x_1 - x_3\rangle$	$\|x_3\rangle$
Add $3\times$ reg. 0 to reg. 1	$\|x_0\rangle$	$\|4x_0 + x_1 + x_2 + x_3\rangle$	$\|2x_1 - x_3\rangle$	$\|x_3\rangle$
Add reg. 2 to $2\times$ reg. 1	$\|x_0\rangle$	$\|8x_0 + 4x_1 + 2x_2 + x_3\rangle$	$\|2x_1 - x_3\rangle$	$\|x_3\rangle$
Add $3\times$ reg. 3 to $2\times$ reg. 2	$\|x_0\rangle$	$\|8x_0 + 4x_1 + 2x_2 + x_3\rangle$	$\|4x_1 + x_3\rangle$	$\|x_3\rangle$
Add $-1\times$ reg. 1 to $2\times$ reg. 2	$\|x_0\rangle$	$\|8x_0 + 4x_1 + 2x_2 + x_3\rangle$	$\|-8x_0 + 4x_1 - 2x_2 + x_3\rangle$	$\|x_3\rangle$
Product on regs. 1 and 2	$\|x_0\rangle$	$\|\boldsymbol{8x_0 + 4x_1 + 2x_2 + x_3}\rangle$	$\|\boldsymbol{-8x_0 + 4x_1 - 2x_2 + x_3}\rangle$	$\|x_3\rangle$
Add reg. 1 to $-1\times$ reg. 2	$\|x_0\rangle$	$\|8x_0 + 4x_1 + 2x_2 + x_3\rangle$	$\|16x_0 + 4x_2\rangle$	$\|x_3\rangle$
Divide reg. 2 by 4	$\|x_0\rangle$	$\|8x_0 + 4x_1 + 2x_2 + x_3\rangle$	$\|4x_0 + x_2\rangle$	$\|x_3\rangle$
Add $6\times$ reg. 2 to reg. 1	$\|x_0\rangle$	$\|32x_0 + 4x_1 + 8x_2 + x_3\rangle$	$\|4x_0 + x_2\rangle$	$\|x_3\rangle$
Add $-15\times$ reg. 0 to $4\times$ reg. 2	$\|x_0\rangle$	$\|32x_0 + 4x_1 + 8x_2 + x_3\rangle$	$\|x_0 + 4x_2\rangle$	$\|x_3\rangle$
Add $15\times$ reg. 3 to reg. 1	$\|x_0\rangle$	$\|32x_0 + 4x_1 + 8x_2 + 16x_3\rangle$	$\|x_0 + 4x_2\rangle$	$\|x_3\rangle$
Divide reg. 1 by 2	$\|x_0\rangle$	$\|16x_0 + 2x_1 + 4x_2 + 8x_3\rangle$	$\|x_0 + 4x_2\rangle$	$\|x_3\rangle$
Add $-15\times$ reg. 0 to reg. 1	$\|x_0\rangle$	$\|x_0 + 2x_1 + 4x_2 + 8x_3\rangle$	$\|x_0 + 4x_2\rangle$	$\|x_3\rangle$
Add reg. 1 to $-2\times$ reg. 2	$\|x_0\rangle$	$\|x_0 + 2x_1 + 4x_2 + 8x_3\rangle$	$\|-x_0 + 2x_1 - 4x_2 + 8x_3\rangle$	$\|x_3\rangle$
Product on regs. 1 and 2	$\|x_0\rangle$	$\|\boldsymbol{x_0 + 2x_1 + 4x_2 + 8x_3}\rangle$	$\|\boldsymbol{-x_0 + 2x_1 - 4x_2 + 8x_3}\rangle$	$\|x_3\rangle$
Add $6\times$ reg. 1 to $-2\times$ reg. 2	$\|x_0\rangle$	$\|x_0 + 2x_1 + 4x_2 + 8x_3\rangle$	$\|8x_0 + 8x_1 + 32x_2 + 32x_3\rangle$	$\|x_3\rangle$
Divide reg. 2 by 8	$\|x_0\rangle$	$\|x_0 + 2x_1 + 4x_2 + 8x_3\rangle$	$\|x_0 + x_1 + 4x_2 + 4x_3\rangle$	$\|x_3\rangle$
Add $-3\times$ reg. 2 to $2\times$ reg. 1	$\|x_0\rangle$	$\|-x_0 + x_1 - 4x_2 + 4x_3\rangle$	$\|x_0 + x_1 + 4x_2 + 4x_3\rangle$	$\|x_3\rangle$
Add $3\times$ reg. 0 to $4\times$ reg. 1	$\|x_0\rangle$	$\|-x_0 + 4x_1 - 16x_2 + 16x_3\rangle$	$\|x_0 + x_1 + 4x_2 + 4x_3\rangle$	$\|x_3\rangle$
Add $12\times$ reg. 3 to reg. 2	$\|x_0\rangle$	$\|-x_0 + 4x_1 - 16x_2 + 16x_3\rangle$	$\|x_0 + x_1 + 4x_2 + 16x_3\rangle$	$\|x_3\rangle$
Add $-3\times$ reg. 0 to $4\times$ reg. 2	$\|x_0\rangle$	$\|-x_0 + 4x_1 - 16x_2 + 16x_3\rangle$	$\|x_0 + 4x_1 + 16x_2 + 64x_3\rangle$	$\|x_3\rangle$
Add $48\times$ reg. 3 to reg. 1	$\|x_0\rangle$	$\|-x_0 + 4x_1 - 16x_2 + 64x_3\rangle$	$\|x_0 + 4x_1 + 16x_2 + 64x_3\rangle$	$\|x_3\rangle$
Product on regs. 1 and 2	$\|x_0\rangle$	$\|\boldsymbol{-x_0 + 4x_1 - 16x_2 + 64x_3}\rangle$	$\|\boldsymbol{x_0 + 4x_1 + 16x_2 + 64x_3}\rangle$	$\|x_3\rangle$
Add reg. 2 to reg. 1	$\|x_0\rangle$	$\|8x_1 + 128x_3\rangle$	$\|x_0 + 4x_1 + 16x_2 + 64x_3\rangle$	$\|x_3\rangle$
Divide reg. 1 by 8	$\|x_0\rangle$	$\|x_1 + 16x_3\rangle$	$\|x_0 + 4x_1 + 16x_2 + 64x_3\rangle$	$\|x_3\rangle$
Add $-4\times$ reg. 1 to reg. 2	$\|x_0\rangle$	$\|x_1 + 16x_3\rangle$	$\|x_0 + 16x_2\rangle$	$\|x_3\rangle$
Add $-1\times$ reg. 0 to reg. 2	$\|x_0\rangle$	$\|x_1 + 16x_3\rangle$	$\|16x_2\rangle$	$\|x_3\rangle$
Divide reg. 2 by 16	$\|x_0\rangle$	$\|x_1 + 16x_3\rangle$	$\|x_2\rangle$	$\|x_3\rangle$
Add $-16\times$ reg. 3 to reg. 1	$\|x_0\rangle$	$\|x_1\rangle$	$\|x_2\rangle$	$\|x_3\rangle$

Table 7.6: $\boldsymbol{k = 4}$ **quantum-quantum multiplication sequence.** In this table only the state of the x sub-registers are shown; the same operations are applied to the y and z registers. Registers containing values upon which the algorithm is applied again recursively are highlighted in bold. The linear combinations used here for the products correspond to the evaluation points $w_i \in \{0, \infty, \pm 1, \pm 1/2, \pm 2, \pm 4\}$.

Theorem 9. *The algorithms recorded in Algs. 7.1 and 7.2, and Tables 7.4, 7.5, and 7.6, can be executed in depth $\mathcal{O}(n^{\log_k p})$ using $\mathcal{O}(n)$ qubits, where p is the number of groups of parallel recursive calls in the algorithm.*

Proof. First we prove the depth bound. If we use fast algorithms for the additions, we may perform them in time $d \log n$ for some constant d [230]. As in the proof of Theorem 6 we denote the number of "overflow" bits of each $(A\boldsymbol{x})_i$ as $t(k)$, a function which does not depend on n. With this we can write a recursion relation

$$D(n) = d \log n + D(n/k + t) \tag{7.42}$$

We note that for any power p, $\mathcal{O}((n/k + t)^p) = \mathcal{O}(n^p) + \mathcal{O}(t^p)$. The second term is $\mathcal{O}(1)$ with respect to n so we may drop it, thus we may solve the recursion with

$$D(n) = \mathcal{O}(n^{\log_k n_p}) \tag{7.43}$$

which is what we desired to show.

Next we show that this depth can be achieved using $\mathcal{O}(n)$ qubits. The logarithmic-depth addition uses $\mathcal{O}(n)$ ancillas which can be reused after the addition is complete. Thus we only need to show that the "overflow" qubits are at most $\mathcal{O}(n)$ across all levels of recursion.

Suppose we cut off the recursion (or at least the parallelism) when the size of input integers is below some constant value n_{cutoff}. At the bottom level of recursion, there will be at most n/n_{cutoff} values being phase multiplied in parallel, each needing up to t overflow ancillas. The number of values being multiplied, and thus the number of overflow ancillas, at each level of the tree of recursive calls is at most half of the level below it; thus by geometric series, the total number of overflow ancillas at all levels is at most twice the number of overflow ancillas at the bottom level. Thus the total number of overflow ancillas is at most $2tn/n_{\text{cutoff}}$, which is linear in n, which is what we desired to show. □

Remark: We note that for the values of k relevant in practice, the number of overflow ancillas t for a single value $(A\boldsymbol{x})_i$ is quite small (see Table 7.2). Since the total number of ancillas needed is bounded by $2nt/n_{\text{cutoff}}$, and n_{cutoff} can be taken to be, say, 50 without drastic loss of parallelism, we expect the qubit usage from "overflow" ancillas to not only be linear in n, but in fact a quite small fraction of n in practice. It is likely that the number of ancillas used for these overflows will be dwarfed by the ancilla usage from sub-linear-time addition.

7.3.5 Fast exact quantum Fourier transform

In this section we show that our results imply a new upper bound for the gate cost of performing an *exact* quantum Fourier transform modulo a power of 2 with only $\mathcal{O}(1)$ ancillas. We note that in practice, the quantum Fourier transform should always be performed approximately, which can be achieved with the standard construction to within error ϵ in time $\mathcal{O}(n \log(1/\epsilon))$. [168] However we believe the following is at least of theoretical interest.

It was shown over two decades ago that the quantum Fourier transform with modulus 2^n, which we may denote QFT_{2^n}, can be implemented exactly via the following three steps for any positive integer $m < n$ (stated here in the language we have been using in this work): [233]

1. Apply $\mathsf{QFT}_{2^{n-m}}$ to the first $n - m$ qubits
2. Apply PhaseProduct($2\pi/2^n$, $|x\rangle$, $|y\rangle$), where x is the value of the first $n - m$ qubits and y is the value of the remaining m qubits
3. Apply QFT_{2^m} to the final m qubits

(Note that $m = 1$ corresponds to the "standard" construction for the QFT). By setting $m = n/2$ and performing steps 1 and 3 via recursive call, the runtime of this algorithm becomes $\mathcal{O}(\log n)$ times the cost of step 2. In the work introducing this construction, it was suggested to implement step 2 by adding a length n ancilla register and computing the product xy in that register, then applying single-qubit phase rotations based on the bits of the product (and then uncomputing the product afterwards). Given our results, we may apply PhaseProduct directly without needing to allocate an extra register. This directly implies that the exact quantum Fourier transform can be implemented in sub-quadratic time without the need for more than the $\mathcal{O}(1)$ ancillas used by PhaseProduct (in fact, just using a single anicilla for the case of $k = 2$).

7.3.6 Practical circuit costs

Having examined the asymptotic costs of our algorithms, we estimate the gate and qubit counts required to implement them in practice. There are a number of tunable parameters which affect the gate counts for different types of gates as well as the qubit counts; here we set the parameters to achieve a particular balance of gate counts and number of ancillas. The results for non-modular multiplication of 2048-bit numbers are summarized in Table 7.7. We find that these algorithms drastically reduce qubit counts, while exhibiting competitive gate counts. Regarding gate counts, we note two things: first, that making a direct comparison between our algorithms and others is not straightforward, because they use different gate sets—for example, a sizeable fraction of the two-qubit gates in our classical-quantum multiplication algorithm are controlled phase rotations, while in other implementations the two- and one-qubit gates might consist entirely of Clifford operations. Second, we note that these estimates for our algorithms correspond to preliminary circuit constructions; it is likely that there are further optimizations to be had in the implementation which could yield considerable improvement in gate counts especially.

For each value of k, we numerically determine the "crossover" point, the value of n above which the algorithm k outperforms $k - 1$. Here, we define "outperform" as using fewer total CCR_ϕ, CR_ϕ, and Toffoli gates. In Table 7.8 we record the optimal crossover points that we find for each k. We note that these crossover points are dependent on the

Algorithm	Gate count (millions)		Ancilla qubits
	Toffoli/CCR_ϕ	2- and 1-qubit	
Classical-quantum multiply-add			
This work	0.46	2.6	12
Digital Windowed [234]	1.8	2.5	4106
Digital Karatsuba [234]	5.6	34	12730
Digital Schoolbook [234]	6.4	38	2048*
Quantum-quantum multiply-add			
This work	54	26	30
Digital Karatsuba† [176]	15	30	13152
Digital Schoolbook† [176]	21	420	4096*

Table 7.7: **Circuit size estimates for multiplication of 2048-bit numbers.** All estimates are in the "abstract circuit model" (no error correction or routing costs included and all qubit counts are logical), thus comparisons between gate counts especially should be considered rough. The "digital" algorithms refer to algorithms which compute the multiplication directly rather than in Fourier space; estimates were computed using the Q# code from [176] and [234] The 2- and 1-qubit gate count column is in addition to the 3-qubit gates; it does not include decomposed 3-qubit gates. *These qubit counts are the values reported in [176] and [234]; we do not see why it would not be possible to perform schoolbook multiplication without the need for ancilla qubits. †The resource estimates for digital quantum-quantum multiplication assume the output register is initialized to zero (that is, they only perform a multiplication, not a "multiply-add").

specific implementation and definition of cost, and so may vary, for example, if the circuits are compiled to target a certain physical gate set. Using these crossover points in our implementation of 2048-bit multiplication, we adjust k throughout the recursion to always use the optimal algorithm for the size of the factors, eventually switching to schoolbook when the factors have become sufficiently small.

For our estimates, we use the Cuccaro quantum adder to perform the additions required by our algorithms [229]. The Cuccaro adder requires one extra ancilla qubit, but it can be reused after the adder is complete. We consider this tradeoff worth it for its low gate counts. For sign inversions and multiplication/division by powers of 2, we note that both can both be implemented "logically" with zero quantum cost. Sign inversion in two's complement representation corresponds to flipping all bits of the integer and incrementing the value by 1. Flipping all bits can be performed logically by relabeling which quantum state corresponds to 0 and which to 1; the addition of one can easily be incorporated into the next addition following the sign flip via the input "carry" bit that is inherently a part of Cuccaro addition.

Algorithm variant	Cutoff
Schoolbook	—
$k = 2$ (Karatsuba)	29
$k = 3$	114

(a) Classical-quantum multiplication

Algorithm variant	Cutoff
Schoolbook	—
$k = 2$	9
$k = 3$	58
$k = 4$	262

(b) Quantum-quantum multiplication

Table 7.8: **Crossover points of the algorithms.** The "cutoff" column lists the smallest value of n for which the given algorithm has a smaller gate count than the one immediately preceding it in the table. The "schoolbook" algorithm is the base case, and thus does not have a cutoff. Note that the values in this table are estimates and depend strongly on the precise implementation of the circuits.

Multiplication and division by powers of 2 correspond just to bit shifts and can be performed for zero quantum cost by relabeling qubits.

Finally, we note that in addition to the cost of the circuits to implement the phases, we must also apply a quantum Fourier transform beforehand and an inverse quantum Fourier transform afterward. If we apply the standard technique of dropping phase rotations below a certain threshold to implement an approximate QFT, the gate count of these operations is negligible (less than the least significant figure listed in Table 7.7) even when the error for the approximate QFT is set as low as 10^{-9}. This standard approximate QFT construction also requires no ancilla qubits.

7.4 Applications and optimizations

In this section we describe various tweaks that can be made to our algorithms to optimize them for two exciting applications: Shor's algorithm, and cryptographic proofs of quantumness. Before discussing those in depth, we begin with an aside about modular multiplication which will be relevant for both applications.

7.4.1 Modular multiplication

So far, we have discussed algorithms for non-modular multiplication—the inputs are each of length n, and the output is of length $2n$. For both the applications we discuss here, we are instead interested in multiplication modulo some integer N. In this section we will show that with a small modification, our algorithms can output the product p in the form of an (approximate) binary fraction $(p \bmod N)/N$ of precision m bits (stored on m qubits), while maintaining their asymptotic costs. Later we will show that this form, with m not much larger than n, is sufficient for relevant applications. Here we discuss only the case

of quantum-quantum multiplication to avoid unnecessarily complicated notation, but the results trivially carry over to the case of classical-quantum multiplication.

In the preceding sections, we have been working in Fourier space modulo 2^{2n}. In Fourier space, the modular multiplication result we desire has the following form (here dropping any constant w to which the result was added, but it can be trivially included):

$$\mathsf{QFT}_{2^{2n}} |xy \bmod N\rangle = \sum_z \exp\left(2\pi i \frac{(xy \bmod N)z}{2^{2n}}\right) |z\rangle \tag{7.44}$$

It is relatively well-known that if we instead work in Fourier space modulo N, we may apply the following identity:

$$\exp\left(2\pi i \frac{(xy \bmod N)z}{N}\right) = \exp\left(2\pi i \frac{xyz}{N}\right) \tag{7.45}$$

Intuitively, as the phase wraps around 2π, it automatically performs the modulo operation for us! This seems to suggest that applying our circuits to modular multiplication is trivial: simply change the modulus of the Fourier transforms.

The challenge is that circuits for performing the quantum Fourier transform with an arbitrary modulus are considerably more complex than for a power of two—and (to our knowledge) there do not exist constructions for the QFT with arbitrary modulus that operate in sub-quadratic time using only $\mathcal{O}(1)$ ancillas, which is what we desire. Instead, we may apply the phase of Eq. 7.45 but perform the final quantum Fourier transform modulo a power of two instead of N. This is simply quantum phase estimation for the phase $2\pi xy/N$. It is well known that quantum phase estimation for a phase that cannot be represented exactly as a binary fraction yields a superposition which is exponentially heavily weighted on the binary strngs that best approximate the value of the phase. [168]

We note that there is another construction by which we can use our circuits to compute modular multiplication *exactly*, with the same asymptotic gate count but now using $3n+\mathcal{O}(1)$ qubits. It is a quantum version of a classical construction called *Montgomery multiplication*. It is particularly nice because classically, Montgomery multiplication has the downside of introducing a (known) extra factor into the result that needs to eventually be dealt with; for classical-quantum multiplication the inverse of this extra factor can be classically multiplied into the classical input, canceling it out and obviating the need to deal with it later. However, we do not pursue that method further here because the approximate version above uses fewer qubits and suffices for our needs.

7.4.2 Skipping unnecessary QFTs

Another optimization relevant for both the applications below involves simplifying quantum Fourier transforms in certain cases. If we know that the "output" register is starting in the $|0\rangle$ state, applying the initial quantum Fourier transform will just yield a uniform superposition over all bitstrings, all with the same phase of $+1$. Thus we can skip the complication of a

Fourier transform modulo 2^m and simply apply a Hadamard gate to each of the qubits in parallel, yielding the same state in depth 1 instead of depth $\mathcal{O}(n)$. (Note that we do still have to perform the full inverse quantum Fourier transform at the end of the multiplication.)

With that, we move on to discussing applications.

7.4.3 Shor's algorithm

The core of Shor's algorithm for factoring an integer of n bits can be implemented via a series of in-place multiplications by classical constants, controlled off of a single qubit (see the "one controlling qubit" trick, Sec. 2.4 of [157]). Written out, a single one of these multiplications consists of the in-place operation

$$|x\rangle \to |cx \bmod N\rangle \tag{7.46}$$

for a classical integer c. Here we apply the modular multiplication introduced in Sec. 7.4.1 to implement this operation to within an error ϵ using $2n+\mathcal{O}(\log 1/\epsilon)$ qubits (while maintaining the same subquadratic gate count of the multiplication algorithm).

To do so, we make the following observation: our classical-quantum multiplication algorithm does not require that the classical value c be an integer. It can be a floating-point number, which we may classically compute to whatever arbitrary precision we desire before using it to compute the values of the phase rotations in our multiplication algorithm. We use this fact to perform the following trick: compute the fractional value $w = (cx \bmod N) \cdot 2^n/N$ up to some precision m as in Eq. 7.45, and then multiply by the value $N/2^n$ to convert the fractional value w into an integer. The accuracy of this operation will depend on the precision to which we compute w; however, we will find that we only need $\mathcal{O}(\log(1/\epsilon))$ extra bits to achieve an error of less than ϵ.

Algorithm 7.5 applies this idea to approximately implement the unitary of Eq. 7.46. It is clear by inspection that it uses $n + \mathcal{O}(\log 1/\epsilon)$ ancilla qubits in time $\mathcal{O}(n^{\log_k q})$. In the following theorem we prove the error bound.

Theorem 10. *The final state $|\psi\rangle$ produced by Algorithm 7.5 has inner product $\langle\psi|cx \bmod N\rangle \geq 1 - \mathcal{O}(\epsilon)$.*

Proof. We begin by showing that if Algorithm 7.5 were run with infinite precision (that is, setting $m \to \infty$), the correct state would result with zero error. Then we show that the difference between the unitary applied in the infinite precision case and the real case is small.

In the infinite precision case, after step 1 we have the state $|x\rangle|w\rangle$ for $w = ((c-1)x \bmod N) \cdot (2^n/N)$. In step 2 we add $w \cdot N/2^n = (c-1)x \mod N$ to x, yielding $cx \bmod N$ or $(cx \bmod N) + 1$ exactly. Step 3 uses an ancilla to reduce this to $cx \bmod N$; the ancilla now contains whether N needed to be subtracted or not. Beauregard observed the identity $(a+b) \bmod N \geq a \Leftrightarrow a + b < N$ [157], in our case this implies that the truth value of the comparison $cx \bmod N < w \cdot N/2^n$ will equal the value of the ancilla and thus can be used to uncompute it. Finally, for step 5, observe that $((1-c^{-1})(cx) \bmod N) = (c-1)x \bmod N = w$

Algorithm 7.5: In-place classical-quantum modular multiplication

Input: Quantum state $|x\rangle$ (extended to superpositions by linearity)
Classical constant c
Error level ϵ

Output: Quantum state $|cx \bmod N\rangle$ (up to error ϵ)

Let $m = n + \lceil 2\log(2 + 1/2\epsilon)\rceil$
Allocate a register of m ancillas initialized to $|0\rangle$

1 Compute $|x\rangle|0\rangle \to |x\rangle|\tilde{w}\rangle$ for $w = ((c-1)x \mod N) \cdot (2^n/N)$ via classical-quantum multiplication, where $\tilde{w}$ is w rounded to a precision of $n + \lceil \log(2 + 1/2\epsilon)\rceil$ bits
2 Compute $|x\rangle|w\rangle \to |x + (N/2^n)w\rangle|w\rangle$. State is now (approximately) $|cx \bmod N\rangle|w\rangle$ or $|(cx \bmod N) + N\rangle|w\rangle$.
3 Using an ancilla qubit, compute whether the left register is greater than N; subtract N controlled by the ancilla. State is now $|cx \bmod N\rangle|w\rangle$
4 Uncompute the ancilla qubit by computing whether $cx \bmod N < w \cdot N/2^n$ via a comparison operator
5 Subtract the value $w = ((1 - c^{-1})(cx) \bmod N)/N$ from the second register, where c^{-1} is the multiplicative inverse of $c \pmod N$. State is now $|cx \bmod N\rangle|0\rangle$.

and thus subtracting it from the second register resets that register to zero. Also note that the multiplicative inverse c^{-1} exists because c and N are co-prime (or, if they're not, $\gcd(c, N)$ provides a factor of N without requiring Shor's algorithm at all!). We now have the state $|cx \bmod N\rangle|0\rangle$ which is what we desired. Now we turn to showing that the approximate circuit, in which $m = n + \mathcal{O}(\log 1/e)$, yields a unitary that is within ϵ of the circuit with $m \to \infty$.

We begin with step 1. Here we must show that applying Hadamard gates to generate $|x\rangle \sum_z |z\rangle$, then a phase rotation of $\phi = 2\pi wz$ and then an inverse QFT module 2^m yields a state that is ϵ close to $|x\rangle|\tilde{w}\rangle$. This can be seen immediately by observing that this process corresponds precisely to the quantum phase estimation algorithm for the phase $2\pi w$, without the usual final measurement to extract an estimation of the phase. It is known that performing quantum phase estimation with an extra $\lceil \log(2 + 1/2\epsilon)\rceil$ qubits beyond the precision to which the phase is desired yields a probability of at least $1 - \epsilon$ of yielding the value of the phase correctly up to the desired precision. [168] Here $\tilde{w}$ corresponds to this "correct" value of the phase which would be observed if the register were measured, so the population of the state $|x\rangle|\tilde{w}\rangle$ must be at least $1 - \epsilon$ and the overall error of this step is at most $\mathcal{O}(\epsilon)$.

Now we move to step 2. Observe that even for the largest value of z, the difference in this phase when w is exact versus truncated to a total of m bits is bounded by $1/2^{m-n}$, and by definition $m - n = \mathcal{O}(\log 1/\epsilon)$. Thus the difference in the phase is $\mathcal{O}(\epsilon)$, and thus the angle between the state resulting from the exact rotation versus the truncated one is proportional

to ϵ.

Steps 3 and 4 are not directly affected by the precision of the second register, so they do not contribute any additional error.

Finally, step 5 simply corresponds to the inverse of step 1. Since the "good" (error-free) portion of the state has $|cx \bmod N\rangle$ in the first register exactly, this step exactly uncomputes the second register.

With that, we see that each individual step has error less than $\mathcal{O}(\epsilon)$ when compared to the case of infinite m; the overall operation's error is bounded by the sum of individual errors and thus is bounded by $\mathcal{O}(\epsilon)$ as well, which is what we desired to show. □

For Shor's algorithm, this operation must be repeated $\mathcal{O}(n)$ times. Using our fast multiplication algorithms of $\mathcal{O}(n^{\log_k(2k-1)})$ time (with $k \geq 2$), this yields an implementation of Shor's algorithm that runs in $\mathcal{O}(n^{1+\log_k(2k-1)})$ time, significantly faster than the usual $\mathcal{O}(n^3)$ runtime for Shor's algorithm. To be more concrete, it is $\mathcal{O}(n^{2.46\cdots})$ for $k = 3$. Importantly, this runtime is achieved while maintaining a very lean qubit count: Algorithm 7.5 uses $2n + \log(1/\epsilon)$ total qubits, and since we must repeat it n times we should set $\epsilon \sim 1/n$. Thus the total qubit count for this implementation of Shor's algorithm is $2n + \mathcal{O}(\log n)$ qubits. To our knowledge this is the first implementation of Shor's algorithm running faster than $\mathcal{O}(n^3)$ time within this space constraint.

Finally we describe an alternative way of implementing Shor's algorithm in fewer than $\mathcal{O}(n^3)$ total gates using our circuits. It takes advantage of the idea of "windowed arithmetic", which has produced some of the fastest known circuits for Shor's algorithm [177], [234]. In brief, instead of multiplying by each classical integer c^b in sequence, a QROM circuit is used to look up the product $\prod c_i^{b_i} \bmod N$ over p values i, from a classical lookup table of size 2^p indexed by the b_i. The QROM circuit puts this value into a quantum register, at which point a quantum-quantum multiplication circuit (which no longer must be controlled by another qubit) can be applied to compute the product. While such a construction reduces the total number of quantum multiplications which need to be performed, they are now quantum-quantum multiplications instead of controlled classical-quantum multiplications; in the context of our circuits it's not clear a priori whether the extra cost (including the cost of the QROM circuit) is worth it. Also, this construction seems to require at least $3n$ qubits, plus the ancilla cost of the QROM. Nevertheless, a valuable direction for further research would be to explicitly count the gates of this construction and compare its cost to the other construction described above.

7.4.4 Proofs of quantumness: $x^2 \bmod N$

As discussed in previous chapters, much recent excitement has focused on creating and implementing efficiently-verifiable "proofs of quantumness," in which an untrusted quantum "prover" can demonstrate its quantum capability to a efficient classical "verifier" [39], [41], [44], [45], [115], [150], [235]. These protocols are interesting not only for their ability to decisively demonstrate quantum computational advantage, but also because they can be mod-

ified to perform more complex quantum cryptographic tasks, such as certifiable generation of quantum random numbers, verifiable remote state preparation, and even the classically-verifiable delegation of arbitrary quantum computations to an untrusted device [39], [47]–[49]. Many of these protocols centrally rely on a cryptographic construction called a trapdoor claw-free function (TCF), and all known TCF constructions involve multiplication of integers in various contexts—making the circuits we develop in this work useful for the implementation of these protocols. In this section we focus on the implementation of the function $f(x) = x^2 \bmod N$ introduced in Chapter 5, which seems to give the best circuit sizes of the currently known TCFs.

To compute $x^2 \bmod N$, we can directly apply our quantum-quantum multiplication circuits, with the inputs x and y being the same register. However, in the specific context of the proof of quantumness introduced in Chapter 5 there are certain further optimizations we can apply which reduce the circuit sizes considerably. With all of these optimizations applied, we find that the proof of quantumness protocols using $x^2 \bmod N$ can be implemented with dramatically fewer intermediate measurements and qubits that in the circuits introduced in Chapter 5, while maintaining competitive gate counts. We also believe more optimized implementations could reduce the gate counts considerably. In Table 7.9 we summarize these costs for various sizes of the modulus N. We now describe these optimizations in detail.

First, we note that the output of the function is measured immediately after it is computed. Thus instead of building a quantum circuit to compute $x^2 \bmod N$ in the form of an integer, we have the freedom to build a circuit that computes any bitstring that uniquely corresponds to such an integer. Classical post-processing can then be used to compute and return the integer value itself. In the context of our circuits, this suggests an immediate optimization: as discussed in Sec. 7.4.1, we can perform the modulo operation automatically in the phase, if we are willing to accept output in the form of a binary fraction representing the value $(x^2 \bmod N)/N$. As long as we have enough bits of precision to uniquely identify an integer from this binary fraction, we may directly measure this output instead of transforming it back to integer form as we did in Shor's algorithm.

Another optimization comes from an observation about the cryptography of trapdoor claw-free functions (TCFs). The classical hardness of the TCF-based "proof of quantumness" protocols comes from the fact that it is hard to find two inputs (x, x') such that $f(x) = f(x')$, where f is a given instance of the TCF family. It is easy to show that this implies that it is also hard to find a pair of values $(g(x'), g(x'))$ for any efficiently-invertible function g—because if such a pair could be found, the function could be immediately inverted to yield (x, x')! Therefore the classical verifier should allow the prover's *input* registers to be modified by the TCF circuit, as long as the prover informs the verifier of the transformation that was made so that the verifier can perform the correct checks on the modified values. This is relevant to our recursive circuits for multiplication, because it means we can drop any gates that serve simply to return the input to its original values. As an example, in the $k = 2$ case, we may allow our circuit to end with the input registers set to $|x_0\rangle\,|x_0 + x_1\rangle$ (resp. z), as long as the verifier knows that this is the value that is being manipulated in the later rounds of the protocol. (In practice we can drop the final gates of *all* levels of recursion, not just

Circuit	Total qubits	Gates (CCR_ϕ/ Toffoli allowed)	Extra mid-circuit measmts.
$n = 128$ (takes seconds on a desktop [174])			
Fast	272	2.3×10^5	0
Narrow	204	4.2×10^5	0
Previous best	942	1.3×10^5	3.4×10^4
$n = 400$ (takes hours on a desktop [174])			
Fast	814	1.8×10^6	0
Narrow	627	3.9×10^6	0
Previous best	3051	8.8×10^5	2.4×10^5
$n = 829$ (record for factoring [175])			
Fast	1676	5.5×10^6	0
Narrow	1268	1.3×10^7	0
Previous best	5522	3.0×10^6	8.0×10^5
$n = 1024$ (exceeds factoring record)			
Fast	2068	7.9×10^6	0
Narrow	1566	1.8×10^7	0
Previous best	6801	4.3×10^6	1.1×10^6

Table 7.9: **Resource estimates for the x^2 mod N portion of a proof-of-quantumness protocol.** These estimates do not include any error correction; depending on the individual gate fidelities it may be possible to mitigate error sufficiently using the post-selection scheme described in Chapter 5. Also note that the additional mid-circuit measurements performed in the cryptographic protocol after x^2 mod N is computed are not included here.

the top level, so the final state will look more complicated—but still be trivially computable by the verifier).

One more benefit comes from the fact that we are performing a square instead of a generic multiplication. A nice feature of the recursive algorithms is that if the top-level operation is to compute a square $x^2 z$, all of the lower levels of recursions will compute squaring operations as well (of the form $(A\boldsymbol{x})_i^2 (A\boldsymbol{z})_i$). Therefore when we eventually reach sufficiently small n in the recursion and switch to the schoolbook algorithm, that operation will be performing a square as well. We note that the complexity of the schoolbook algorithm for applying a phase proportional to $x^2 z$ is roughly half the cost of the same operation for xyz, because rotations of the form $x_i x_j z_k$ and $x_j x_i z_k$ are identical and can be combined. Also, phase rotations of the form $x_i x_i z_k$ (in which both x bits are the same) correspond to a single-controlled phase

rotation, instead of a doubly-controlled one. These facts yield to a significant reduction in the cost of performing the controlled-phase rotations.

If space is very limited, yet another optimization stems from the fact that we are measuring the output immediately as it is produced. Instead of computing all the bits of the output at once, we may instead compute subsets of the bits of the output sequentially. For example, one may let the output register be of length only roughly $n/2$, and perform a smaller quantum-quantum multiplication to compute the less-significant half of the output and measure it. Then the same process can be repeated for the more-significant half, re-using the same output qubits. This optimization reduces the total qubit count from $2n + \mathcal{O}(1)$ to $1.5n + \mathcal{O}(1)$. The gate count will be somewhat larger due to the fact that the recursive multiplication cannot be applied to the full output at once, but by making the decomposition $x^2 = 2^n x_0^2 + 2^{n/2+1} x_0 x_1 + x_1^2$ we note that gate cost can be reduced to that of 6 quantum-quantum multiplications of size $n/2$ (as opposed to one quantum-quantum multiplication of size n). This type of optimization can be extended to reduce the qubit count as low as $n + \mathcal{O}(1)$, but we believe that taking it any further than just described in practice is not likely to be worth the tradeoff in gate count.

Finally we note that there is another optimization, described when the "computational Bell test" protocol was originally introduced in Chapter 5, in which uncomputation of "garbage bits" can be replaced simply by measurement in the Hadamard basis. For the circuits we introduce here, this optimization is not actually very helpful, because there are only $\mathcal{O}(1)$ garbage bits introduced in the circuit (the overflow bits of each $(A\boldsymbol{x})_i$, resp. z)! In fact, we estimate that the experimental cost of performing extra intermediate measurements instead of the $\mathcal{O}(1)$ gates to uncompute these values directly is almost certainly not worth it. We consider a serious benefit of the circuits introduced here that they require no extra intermediate measurement at all (except the measurements explicitly required by the protocol).

7.5 Discussion and outlook

Developing quantum circuits based on classical algorithms is a subtle pursuit. Peculiarities of the quantum setting, such as the need for reversibility and the limited number of qubits on near- and medium-term quantum devices, preclude the straightforward translation of classical circuits to quantum ones, especially for recursive algorithms. However, quantum mechanics also provides us with new computational techniques that do not have classical analogues. In this work we combine the algorithmic speedups from recursive fast multiplication algorithms, which are widely used in classical computing, with the inherently quantum technique of performing arithmetic in the phase of a quantum state and then using a quantum Fourier transform to recover the result. We show that this yields circuits for performing multiplication of numbers stored in quantum registers in a number of gates that is sub-quadratic in the size of the inputs, for both classical-quantum and quantum-quantum multiplication. This gate count scaling is achieved while using an extremely small number of qubits—as

low as just two ancillas in addition to the registers storing the input and output. Our algorithms also offload a considerable portion of the work to classical pre-computation in the circuit compilation phase, significantly reducing the overhead of these asymtotically-faster algorithms. We show that this reduction of overhead makes the algorithms not just a theoretical novelty, but potentially the smallest known circuits for the multiplication of integers at sizes relevant in practice, for computations such as Shor's algorithm and cryptographic protocols for demonstrating quantum advantage.

There are several potential optimizations to our algorithm that warrant further exploration—although there exist subtle pitfalls which must be considered along the way. One open question is whether the techniques presented here can be applied to fast Fourier transform based classical multiplication algorithms, the foremost being the Schonhage-Strassen algorithm [182]. This algorithm replaces the matrix-vector products in, for example, $A^{-1}(A\boldsymbol{x} \circ A\boldsymbol{z})$ with *discrete Fourier transforms*, which can be computed in $\mathcal{O}(n \log n)$ time! (The discrete Fourier transform here should not be confused with the quantum Fourier transform already present in our algorithms—implementing Schonhage-Strassen on a quantum computer would correspond to implementing a quantum circuit for a *classical* fast Fourier transform algorithm, that outputs bitstrings representing the classical discrete Fourier transform of the input). The subtle but seemingly inescapable obstacle preventing this from fitting directly into our framework is that in Schonhage-Strassen, A, $\boldsymbol{x}$, and $\boldsymbol{z}$ are defined on a ring with some modulus (different from the modulus N we discuss in Sec. 7.4.1), while the vector $\boldsymbol{e}$ against which the results are multiplied during the recomposition step is defined over all integers. Because they are defined over different spaces, the matrix-vector products are no longer associative—so precomputing $\boldsymbol{e}^\intercal A^{-1}$ will no longer yield the correct answer! Another outstanding question is whether a quantum circuit for the discrete Fourier transform can be devised that uses only a constant (or even logarithmic) number of ancilla qubits.

Within the algorithms that we do present, we believe there is still much room for optimization. The fact that the phase product portion of our circuits can be computed in sub-linear depth begs the question of whether there is a corresponding sub-linear depth algorithm for the quantum Fourier transform, which uses only a linear number of qubits. If such an algorithm were applied to our circuits, the entire multiplication could be performed in sub-linear depth. We also note that we do not explicitly give a sequence of additions and subtractions that yield a sub-linear depth circuit for quantum-quantum multiplication. For $k = 4$ PhaseTripleProduct, computing the 10 sub-products in three parallel groups would yield a sub-linear depth circuit; the challenge is coming up with a sequence of arithmetic operations that yields the appropriate linear combinations at the right times. We are virtually certain that such a sequence of operations can be devised; we have not explored this extensively due to the orthogonal issue of the lack of a sub-linear quantum Fourier transform. More broadly, in all our algorithms it is worth exploring if there are better sequences that yield the right linear combinations in fewer addition operations that the sequences we propose. Perhaps there is an automated or algorithmic way to come up with these optimal sequences in the general case.

Finally we note that there are certain optimizations applicable in the "schoolbook" Fourier

multiplication algorithm that don't translate in an obvious way to our algorithms. [156], [157] For example, a classic optimization is to drop very small rotations that have a negligible effect on the state. Unfortunately in our case there seem to be very few rotations small enough to be dropped, because the elements of $\hat{\boldsymbol{e}}$ are complicated linear combinations of various powers of two—and thus they usually contain at least one large term. One seemingly clever idea when using our circuits in the proofs of quantumness described in Section 7.4 would be to set the modulus N so that one or more of the elements of $\hat{\boldsymbol{e}}$ are close to a multiple of N. Then for those values, $\hat{e}_i \bmod N$ would be very small and whole branches of the recursive tree could be ignored. Unfortunately there is an extremely subtle but critical danger in doing this: the elements of $\hat{\boldsymbol{e}}$ have low hamming weight (number of bits set to 1), and thus so would N. The classical special number field sieve algorithm is much more efficient at factoring numbers with low Hamming weight than numbers of a general form, so this technique would likely destroy the classical hardness of the quantum advantage protocol! [236]

Finally, we note that in this work we leave largely unexplored a number of practical considerations, such as how this algorithm interacts with error correction, qubit routing, and decomposition to native gate sets. We believe that these questions are important to explore, because our estimates in the abstract circuit model suggest that our algorithms could meaningfully reduce the resources required for quantum computations with real-world impacts and applications. This is especially true if future works find further optimizations (which we believe are likely to exist). Taking a broader perspective, the techniques for multiplication of quantum integers presented here represent a fundamentally new direction for the implementation of quantum arithmetic, combining two ideas which previously seemed ill-suited to working together. Instead, we find that their combination yields the dramatic benefits of both.

Chapter 8

Conclusion

"Rapid calculation, all right," went on the inventor. "It has to try out in a certain formula about ten million numbers. Each number would take a man at least six minutes to examine, which comes to sixty million minutes, or about a million hours. A man could not work at this sort of thing more than ten hours a day, so that gives a hundred thousand days. One could do it in three hundred years if he did not get stale." "How fast is the machine working on this list of ten million numbers?" some one asked. "About a hundred thousand a minute," replied the young man. "It may take an hour and a half to clean up the problem. With a larger driving motor we could make it in twenty minutes. The electric eye would catch it if it were going five times as fast."

Suddenly, click! The power was shut off. "It must have seen something." The machine was turned slowly back till a tell-tale beam of light appeared through the little hole before which the electric eye had been watching. Then some reading of dials and a little grinding of a computing machine and two numbers were found such that the square of one of them plus seven times the square of the other were equal to the number under examination. ...

The machine had done its duty. ... A few minutes computation still remained, and thus it was, while coffee was being served on one of the working tables in the laboratory the big number was broken up into the factors 59,957 and 88,114,244,437. These are the two hidden numbers which when multiplied together will give the sixteen digit number under examination. It may seem to the man in the street an odd thing to get excited about, but on this occasion

All Rome sent forth a rapturous cry,
And even the ranks of Tuscany
Could scarce forbear to cheer.

— Derrick N. Lehmer, *Hunting Big Game in the Theory of Numbers* (1932)

The problem of integer factorization has come up repeatedly throughout this dissertation, so I hope that the reader will forgive me for beginning the conclusion with a somewhat silly anecdote about it. Long before Shor's algorithm, and long before even the RSA cryptosystem based on factoring, number theorists had a strong interest in factoring integers. It is such a straightforward problem—the inverse of multiplication, an operation which has been known since time immemorial—yet for so long has remained stubbornly inefficient. Before the turn of the 20^{th} century, the endeavor of factoring numbers essentially consisted of coming up with successively more and more clever methods of doing so by hand. One such method involved the realization that factoring N could be accomplished by choosing a few small numbers p_i (say, $13, 17, 19, 23, 25$), and finding an integer x for which $x \bmod p_i$ (the remainder when divided by each of the small numbers) landed on one of a few "good" values, for all p_i simultaneously.[1] Of course, simply exhaustively searching through integers by hand in an effort to find such an x is extremely slow. So, in the late 1920s, Derrick H. Lehmer went into the student shop of the physics building at UC Berkeley (precisely the same building in which the work presented in this dissertation was performed!) and built a device consisting of an axle with several gears on it, a loop of bicycle chain hanging off of each gear, and some basic electrical components. [238] The number of links in each bicycle chain was set to correspond to each p_i, and a small piece of metal was attached to the chain links that corresponded to the "good" set of values modulo p_i. As the axle rotated, it incremented a counter mechanism that displayed a value x; the bike chain loops, having length p_i, naturally tracked the values of $x \bmod p_i$. The device was designed such that when the attached metal pieces all *simultaneously* aligned—corresponding to an $x \bmod p_i$ in the "good" set for all p_i—it completed an electric circuit causing the axle's rotation to stop, revealing x on the counter and, with an easy further calculation, the factors of N. (The story recounted in the quote from Lehmer above, with its "electric eye," corresponds to a later iteration of the machine which used a photoelectric detector as a more reliable way to detect when to stop the axle. Despite the fantastical-sounding nature of it, as far as I know it is a real story. If only it were still acceptable to write scientific papers with such flowery language...) It is not an understatement to say this machine *revolutionized* the practice of factoring numbers. It smashed records for the largest numbers that had been factored at that time by orders of magnitude, and kicked off a new era of using machines, rather than pen and paper, to tackle this, and other, challenging problems.

At some level I feel that despite the dramatic difference in the hardware available to us, modern computing research, and especially quantum computing, is in the same spirit as Lehmer's work. As researchers in these fields, our task is to take the physical laws of the world in which we live—making use of the inventions of those who came before us, whether they be bicycle chains or semiconductors—and use them to process information. I am honored to have had the chance, in this dissertation work, to contribute small steps forward in this endeavor, alongside many colleagues all across the world.

[1] The set of "good" values for each p_i is a function of $N \bmod p_i$. For a nice overview of how this works, see [237] or any article on the "Lehmer sieve."

Of course, the work continues. For the reasons laid out in Section 1.1 of the introduction, I expect classical numerical study to remain an important tool in the analysis and development of quantum systems for years to come. Classical computing hardware is advancing rapidly, and leveraging new technologies—whether GPU acceleration as discussed in Chapter 2, or perhaps even newer types of processors just appearing on the horizon such as tensor processing units (TPUs) [88], [239]—will require constantly upgrading the software implementing our numerical techniques. As shown in Chapter 3, tuning numerical techniques for the specific problem at hand has the potential to yield considerable benefits; it would be interesting to explore which of those innovations can applied more generally, or at least ported to more general software packages like the `dynamite` library presented in Chapter 2. One specific direction which has already begun to be explored (in yet unpublished work) is the potential to apply the techniques of Chapter 3 to new quantum systems with similar structure, such as the Heisenberg plus random field model in two dimensions.

In the field of quantum cryptographic protocols, there are too many open research directions to list them all here. The most direct extension of the work in Part II is to continue the push to implement an efficiently-verifiable demonstration of quantum advantage *at scale*. One idea which to my knowledge has not been explored much is to relax our definition of "efficient" verification. The quantum advantage protocols discussed in Part II of this dissertation can be verified by a classical machine in polynomial-time—but this may be overkill. A protocol which is exponentially hard to reproduce classically, and also exponentially hard to verify, could still be useful if the verification exponential is considerably *smaller*. For example, if reproducing the results takes 2^n classical operations, but verifying them takes only $2^{n/2}$, it would be possible to run an experiment that is classically infeasible to reproduce but still can be verified with some effort.

Taking a broader perspective, demonstrations of quantum advantage will only remain interesting for some time—after efficiently-verifiable ones have been convincingly implemented, they will not have much direct use. But quantum cryptographic protocols with similar structure have already been shown to be useful for a number of more interesting tasks, such as certifiable randomness generation, verifiable remote state preparation, and verifiable delegated quantum computing. [39], [47], [48] A clear next step forward is to explore other types of practically-useful tasks to which these protocols may be applied.

Looking at the field of quantum computing more generally, it is up to anyone's guess what the future holds. From the theoretical side, there are few explicit proofs bounding how things may go. The number of problems for which a truly radical superpolynomial quantum speedup has been explicitly shown is not large—and this set consists largely of somewhat abstract number theoretic problems, which are only considered important to people's lives due to their use in cryptography. Furthermore, from the perspective of complexity theory, there is not even any real evidence that factoring, for example, is a hard problem classically: our best evidence so far is that a lot of very smart people have tried to figure out how to do it efficiently, and nobody has succeeded. On a somewhat similar note, quantum computers are widely conjectured (or hoped, depending on your perspective) to be useful for crucial real-world applications such as quantum chemistry—but concrete evidence for these claims

is hard to come by. [240] Given these points, I see three broad paths that the future of quantum computing may follow.

In one (perhaps unlikely, but I think not implausible) future, an efficient *classical* algorithm (or algorithms) will be discovered for the problems like factoring that quantum computers have been conjectured to dramatically to speed up. This is the world in which the complexity classes BQP and BPP—informally, the problems efficiently solvable by quantum and classical computers respectively—are equal. In this case we would be stuck with only polynomial speedups from quantum computers, and getting any useful advantage from them in solving real-world problems would require astounding advances in quantum hardware (see Section 1.2 of the introduction). Hopefully such hardware advances will be achieved, but it seems likely to take many, many years of work. In this world, quantum computers can still find use as extremely precisely programmable physics experiments, with the ability to implement large classes of quantum Hamiltonians with the push of a button.

In another future, problems like factoring remain hard classically, but no new "killer" applications are found beyond the broad categories of quantum speedup already explicitly known. In this case, quantum computers will eventually force the world's cryptography to move to post-quantum secure algorithms, but aside from that, the outlook doesn't actually look much different than that of the BQP=BPP world. Building quantum computers applicable to other real-world problems will take an enormous amount of effort.

In the third and most optimistic future, new applications of quantum computers, in which they show considerable and practical advantage over classical ones, will be discovered in the next few years. It seems that this future is the one that many people are hoping for (and in some cases betting their money on). It is not a terrifically implausible scenario: as discussed in the introduction, it is very difficult to discover new algorithms if you can only run them in your imagination. Hopefully, with the increasing availability of programmable quantum systems of considerable size, exploring their capabilities will lead to unexpected new directions. Such a situation would not be without precedent: an example directly relevant to Chapter 2 is the case of classical Krylov subspace algorithms for the matrix exponential. To quote Sidje [78], "It seems these techniques were long established among chemical physicists without much justification other than their satisfactory behavior." That is, someone just *tried it*, and it seemed to work pretty well! Only later was it rigorously understood why the error remained well-bounded. Perhaps new quantum algorithms will follow this path as well, with meaningful impacts on the world's gravest problems such as climate change.

With that, I would like to conclude with a reminder that we ought to consider not only what problems technologies like quantum computing could solve, but also *whether they are the right solution to pursue*. Take climate change, for example: as just discussed, there is a small chance that quantum computers will lead to a breakthrough in carbon capture or, say, the production of fertilizer, that meaningfully helps to avert climate catastrophe. But the thing is, we already *know* how to avert climate catastrophe—by cutting down on the many wildly inefficient and unnecessary sources of greenhouse gas emissions that permeate our society. We even already know how to capture carbon—by preserving and restoring the world's forest and ocean ecosystems which serve as massive natural carbon sinks. To be

clear, I am not suggesting that investment in technology like quantum computing is useless. I certainly think it is worth pursuing, and I care on a personal level about its advancement. I simply hope that we can move it forward without ignoring the other, potentially less flashy, yet very important solutions to the critical problems of our world.

References

[1] D. Aharonov and U. V. Vazirani, "Is Quantum Mechanics Falsifiable? A Computational Perspective on the Foundations of Quantum Mechanics," en, in *Computability: Turing, Gödel, Church, and Beyond*, MIT Press, 2013, pp. 329–349, ISBN: 978-0-262-31267-7, DOI: 10.7551/mitpress/8009.003.0012.

[2] P. W. Shor, *The Early Days of Quantum Computation*, arXiv:2208.09964 [quant-ph], Oct. 2022, DOI: 10.48550/arXiv.2208.09964.

[3] J. Preskill, *Quantum computing 40 years later*, arXiv:2106.10522 [quant-ph], Feb. 2023, DOI: 10.48550/arXiv.2106.10522.

[4] E. Martín-López, A. Laing, T. Lawson, R. Alvarez, X.-Q. Zhou, and J. L. O'Brien, "Experimental realization of Shor's quantum factoring algorithm using qubit recycling," en, *Nature Photonics*, vol. 6, no. 11, pp. 773–776, Nov. 2012, Number: 11 Publisher: Nature Publishing Group, ISSN: 1749-4893, DOI: 10.1038/nphoton.2012.259.

[5] J. A. Smolin, G. Smith, and A. Vargo, "Oversimplifying quantum factoring," en, *Nature*, vol. 499, no. 7457, pp. 163–165, Jul. 2013, Number: 7457 Publisher: Nature Publishing Group, ISSN: 1476-4687, DOI: 10.1038/nature12290.

[6] D. Gottesman, *The Heisenberg Representation of Quantum Computers*, arXiv:quant-ph/9807006, Jul. 1998, DOI: 10.48550/arXiv.quant-ph/9807006.

[7] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits," *Physical Review A*, vol. 70, no. 5, p. 052 328, Nov. 2004, Publisher: American Physical Society, DOI: 10.1103/PhysRevA.70.052328.

[8] G. Carleo and M. Troyer, "Solving the quantum many-body problem with artificial neural networks," *Science*, vol. 355, no. 6325, pp. 602–606, Feb. 2017, Publisher: American Association for the Advancement of Science, DOI: 10.1126/science.aag2302.

[9] D. R. Vivas, J. Madroñero, V. Bucheli, L. O. Gómez, and J. H. Reina, *Neural-Network Quantum States: A Systematic Review*, arXiv:2204.12966 [quant-ph], Apr. 2022, DOI: 10.48550/arXiv.2204.12966.

[10] A. J. Daley, "Quantum trajectories and open many-body quantum systems," *Advances in Physics*, vol. 63, no. 2, pp. 77–149, Mar. 2014, Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00018732.2014.933502, ISSN: 0001-8732, DOI: 10.1080/00018732.2014.933502.

[11] R. Orús, "A practical introduction to tensor networks: Matrix product states and projected entangled pair states," en, *Annals of Physics*, vol. 349, pp. 117–158, Oct. 2014, ISSN: 0003-4916, DOI: 10.1016/j.aop.2014.06.013.

[12] R. Orús, "Tensor networks for complex quantum systems," en, *Nature Reviews Physics*, vol. 1, no. 9, pp. 538–550, Sep. 2019, Number: 9 Publisher: Nature Publishing Group, ISSN: 2522-5820, DOI: 10.1038/s42254-019-0086-7.

[13] J. I. Cirac, D. Pérez-García, N. Schuch, and F. Verstraete, "Matrix product states and projected entangled pair states: Concepts, symmetries, theorems," *Reviews of Modern Physics*, vol. 93, no. 4, p. 045 003, Dec. 2021, Publisher: American Physical Society, DOI: 10.1103/RevModPhys.93.045003.

[14] G. Vidal, "Entanglement Renormalization," *Physical Review Letters*, vol. 99, no. 22, p. 220 405, Nov. 2007, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.99.220405.

[15] S. Aaronson and A. Arkhipov, "The computational complexity of linear optics," in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, ser. STOC '11, New York, NY, USA: Association for Computing Machinery, Jun. 2011, pp. 333–342, ISBN: 978-1-4503-0691-1, DOI: 10.1145/1993636.1993682.

[16] M. J. Bremner, A. Montanaro, and D. J. Shepherd, "Average-Case Complexity Versus Approximate Simulation of Commuting Quantum Computations," *Physical Review Letters*, vol. 117, no. 8, p. 080 501, Aug. 2016, DOI: 10.1103/PhysRevLett.117.080501.

[17] A. P. Lund, M. J. Bremner, and T. C. Ralph, "Quantum sampling problems, BosonSampling and quantum supremacy," en, *npj Quantum Information*, vol. 3, no. 1, pp. 1–8, Apr. 2017, Number: 1 Publisher: Nature Publishing Group, ISSN: 2056-6387, DOI: 10.1038/s41534-017-0018-2.

[18] A. W. Harrow and A. Montanaro, "Quantum computational supremacy," en, *Nature*, vol. 549, no. 7671, pp. 203–209, Sep. 2017, Number: 7671 Publisher: Nature Publishing Group, ISSN: 1476-4687, DOI: 10.1038/nature23458.

[19] B. M. Terhal, "Quantum supremacy, here we come," en, *Nature Physics*, vol. 14, no. 6, pp. 530–531, Jun. 2018, Number: 6 Publisher: Nature Publishing Group, ISSN: 1745-2481, DOI: 10.1038/s41567-018-0131-y.

[20] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, *et al.*, "Characterizing quantum supremacy in near-term devices," en, *Nature Physics*, vol. 14, no. 6, pp. 595–600, Jun. 2018, Number: 6 Publisher: Nature Publishing Group, ISSN: 1745-2481, DOI: 10.1038/s41567-018-0124-x.

[21] A. Bouland, B. Fefferman, C. Nirkhe, and U. Vazirani, "On the complexity and verification of quantum random circuit sampling," en, *Nature Physics*, vol. 15, no. 2, pp. 159–163, Feb. 2019, Number: 2 Publisher: Nature Publishing Group, ISSN: 1745-2481, DOI: 10.1038/s41567-018-0318-2.

[22] S. Aaronson and L. Chen, "Complexity-theoretic foundations of quantum supremacy experiments," in *Proceedings of the 32nd Computational Complexity Conference*, ser. CCC '17, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Jul. 2017, 22:1–22:67, ISBN: 978-3-95977-040-8, (visited on 01/18/2023).

[23] C. Neill, P. Roushan, K. Kechedzhi, *et al.*, "A blueprint for demonstrating quantum supremacy with superconducting qubits," en, *Science*, vol. 360, no. 6385, pp. 195–199, Apr. 2018, Publisher: American Association for the Advancement of Science Section: Report, ISSN: 0036-8075, 1095-9203, DOI: 10.1126/science.aao4309.

[24] C. Huang, F. Zhang, M. Newman, *et al.*, "Classical Simulation of Quantum Supremacy Circuits," *arXiv:2005.06787 [quant-ph]*, May 2020, arXiv: 2005.06787, [Online]. Available: http://arxiv.org/abs/2005.06787 (visited on 04/15/2022).

[25] F. Pan and P. Zhang, "Simulation of Quantum Circuits Using the Big-Batch Tensor Network Method," *Physical Review Letters*, vol. 128, no. 3, p. 030 501, Jan. 2022, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.128.030501.

[26] J. Gray and S. Kourtis, "Hyper-optimized tensor network contraction," *Quantum*, vol. 5, p. 410, Mar. 2021, arXiv: 2002.01935, ISSN: 2521-327X, DOI: 10.22331/q-2021-03-15-410.

[27] F. Pan, K. Chen, and P. Zhang, "Solving the Sampling Problem of the Sycamore Quantum Circuits," *Physical Review Letters*, vol. 129, no. 9, p. 090 502, Aug. 2022, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.129.090502.

[28] Y. (Liu, X. (Liu, F. (Li, *et al.*, "Closing the "quantum supremacy" gap: Achieving real-time simulation of a random quantum circuit using a new Sunway supercomputer," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21, New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 1–12, ISBN: 978-1-4503-8442-1, DOI: 10.1145/3458817.3487399.

[29] X. Liu, C. Guo, Y. Liu, *et al.*, "Redefining the Quantum Supremacy Baseline With a New Generation Sunway Supercomputer," *arXiv:2111.01066 [quant-ph]*, Nov. 2021, arXiv: 2111.01066, [Online]. Available: http://arxiv.org/abs/2111.01066 (visited on 04/15/2022).

[30] X. Gao, M. Kalinowski, C.-N. Chou, M. D. Lukin, B. Barak, and S. Choi, "Limitations of Linear Cross-Entropy as a Measure for Quantum Advantage," *arXiv:2112.01657 [cond-mat, physics:quant-ph]*, Dec. 2021, arXiv: 2112.01657, [Online]. Available: http://arxiv.org/abs/2112.01657 (visited on 04/15/2022).

[31] D. Aharonov, X. Gao, Z. Landau, Y. Liu, and U. Vazirani, "A Polynomial-Time Classical Algorithm for Noisy Random Circuit Sampling," in *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, ser. STOC 2023, New York, NY, USA: Association for Computing Machinery, Jun. 2023, pp. 945–957, ISBN: 978-1-4503-9913-5, DOI: 10.1145/3564246.3585234.

[32] F. Arute, K. Arya, R. Babbush, *et al.*, "Quantum supremacy using a programmable superconducting processor," en, *Nature*, vol. 574, no. 7779, pp. 505–510, Oct. 2019, ISSN: 1476-4687, DOI: 10.1038/s41586-019-1666-5.

[33] H.-S. Zhong, H. Wang, Y.-H. Deng, *et al.*, "Quantum computational advantage using photons," en, *Science*, vol. 370, no. 6523, pp. 1460–1463, Dec. 2020, Publisher: American Association for the Advancement of Science Section: Report, ISSN: 0036-8075, 1095-9203, DOI: 10.1126/science.abe8770.

[34] Y. Wu, W.-S. Bao, S. Cao, *et al.*, "Strong Quantum Computational Advantage Using a Superconducting Quantum Processor," *Physical Review Letters*, vol. 127, no. 18, p. 180 501, Oct. 2021, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.127.180501.

[35] Q. Zhu, S. Cao, F. Chen, *et al.*, "Quantum computational advantage via 60-qubit 24-cycle random circuit sampling," en, *Science Bulletin*, vol. 67, no. 3, pp. 240–245, Feb. 2022, ISSN: 2095-9273, DOI: 10.1016/j.scib.2021.10.017.

[36] M. J. Bremner, R. Jozsa, and D. J. Shepherd, "Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 467, no. 2126, pp. 459–472, Feb. 2011, DOI: 10.1098/rspa.2010.0301.

[37] X. Chen, B. Cheng, Z. Li, *et al.*, "Experimental cryptographic verification for near-term quantum cloud computing," en, *Science Bulletin*, vol. 66, no. 1, pp. 23–28, Jan. 2021, ISSN: 2095-9273, DOI: 10.1016/j.scib.2020.08.013.

[38] D. Chaum, J.-H. Evertse, and J. van de Graaf, "An Improved Protocol for Demonstrating Possession of Discrete Logarithms and Some Generalizations," en, in *Advances in Cryptology — EUROCRYPT' 87*, D. Chaum and W. L. Price, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 1988, pp. 127–141, ISBN: 978-3-540-39118-0, DOI: 10.1007/3-540-39118-5_13.

[39] Z. Brakerski, P. Christiano, U. Mahadev, U. Vazirani, and T. Vidick, "A Cryptographic Test of Quantumness and Certifiable Randomness from a Single Quantum Device," EN, *Journal of the ACM (JACM)*, Aug. 2021, Publisher: ACM PUB27 New York, NY, DOI: 10.1145/3441309.

[40] N. Alamati, G. Malavolta, and A. Rahimi, "Candidate Trapdoor Claw-Free Functions from Group Actions with Applications to Quantum Protocols," en, in *Theory of Cryptography*, E. Kiltz and V. Vaikuntanathan, Eds., ser. Lecture Notes in Computer Science, Cham: Springer Nature Switzerland, 2022, pp. 266–293, ISBN: 978-3-031-22318-1, DOI: 10.1007/978-3-031-22318-1_10.

[41] Z. Brakerski, V. Koppula, U. Vazirani, and T. Vidick, "Simpler Proofs of Quantumness," in *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*, S. T. Flammia, Ed., ser. Leibniz International Proceedings in Informatics (LIPIcs), ISSN: 1868-8969, vol. 158, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 8:1–8:14, ISBN: 978-3-95977-146-7, DOI: 10.4230/LIPIcs.TQC.2020.8.

[42] R. Canetti, O. Goldreich, and S. Halevi, "The random oracle methodology, revisited," *Journal of the ACM*, vol. 51, no. 4, pp. 557–594, Jul. 2004, ISSN: 0004-5411, DOI: 10.1145/1008731.1008734.

[43] N. Koblitz and A. J. Menezes, "The random oracle model: A twenty-year retrospective," en, *Designs, Codes and Cryptography*, vol. 77, no. 2, pp. 587–610, Dec. 2015, ISSN: 1573-7586, DOI: 10.1007/s10623-015-0094-2.

[44] T. Yamakawa and M. Zhandry, "Verifiable Quantum Advantage without Structure," in *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, ISSN: 2575-8454, Oct. 2022, pp. 69–74, DOI: 10.1109/FOCS54457.2022.00014.

[45] Y. Kalai, A. Lombardi, V. Vaikuntanathan, and L. Yang, *Quantum Advantage from Any Non-Local Game*, arXiv:2203.15877 [quant-ph], Mar. 2022, DOI: 10.48550/arXiv.2203.15877.

[46] U. Mahadev, "Classical Homomorphic Encryption for Quantum Circuits," *SIAM Journal on Computing*, FOCS18–189, Dec. 2020, Num Pages: FOCS18-215 Publisher: Society for Industrial and Applied Mathematics, ISSN: 0097-5397, DOI: 10.1137/18M1231055.

[47] U. Mahadev, "Classical Verification of Quantum Computations," in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, ISSN: 2575-8454, Oct. 2018, pp. 259–267, DOI: 10.1109/FOCS.2018.00033.

[48] A. Gheorghiu and T. Vidick, "Computationally-Secure and Composable Remote State Preparation," in *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, ISSN: 2575-8454, Nov. 2019, pp. 1024–1033, DOI: 10.1109/FOCS.2019.00066.

[49] Z. Brakerski, A. Gheorghiu, G. D. Kahanamoku-Meyer, E. Porat, and T. Vidick, *Simple Tests of Quantumness Also Certify Qubits*, arXiv:2303.01293 [quant-ph], Mar. 2023, DOI: 10.48550/arXiv.2303.01293.

[50] M. Fishman, S. R. White, and E. M. Stoudenmire, "The ITensor Software Library for Tensor Network Calculations," *SciPost Phys. Codebases*, p. 4, 2022, Publisher: SciPost, DOI: 10.21468/SciPostPhysCodeb.4.

[51] J. Hauschild and F. Pollmann, "Efficient numerical simulations with Tensor Networks: Tensor Network Python (TeNPy)," *SciPost Phys. Lect. Notes*, p. 5, 2018, Publisher: SciPost _eprint: 1805.00055, DOI: 10.21468/SciPostPhysLectNotes.5.

[52] P. Weinberg and M. Bukov, "QuSpin: A Python package for dynamics and exact diagonalisation of quantum many body systems part I: Spin chains," *SciPost Phys.*, vol. 2, p. 003, 2017, Publisher: SciPost, DOI: 10.21468/SciPostPhys.2.1.003.

[53] J. Schulenburg, *SPINPACK*, [Online]. Available: https://www-e.uni-magdeburg.de/jschulen/spin/ (visited on 04/26/2023).

[54] T. Westerhout, *SpinED*, original-date: 2020-10-11T22:42:04Z, Apr. 2023, [Online]. Available: https://github.com/twesterhout/spin-ed (visited on 04/26/2023).

[55] T. Mori, T. Kuwahara, and K. Saito, "Rigorous Bound on Energy Absorption and Generic Relaxation in Periodically Driven Quantum Systems," *Physical Review Letters*, vol. 116, no. 12, p. 120 401, Mar. 2016, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.116.120401.

[56] T. Kuwahara, T. Mori, and K. Saito, "Floquet–Magnus theory and generic transient dynamics in periodically driven many-body quantum systems," en, *Annals of Physics*, vol. 367, pp. 96–124, Apr. 2016, ISSN: 0003-4916, DOI: 10.1016/j.aop.2016.01.012.

[57] D. A. Abanin, W. De Roeck, W. W. Ho, and F. Huveneers, "Effective Hamiltonians, prethermalization, and slow energy absorption in periodically driven many-body systems," *Physical Review B*, vol. 95, no. 1, p. 014 112, Jan. 2017, Publisher: American Physical Society, DOI: 10.1103/PhysRevB.95.014112.

[58] D. V. Else, B. Bauer, and C. Nayak, "Prethermal Phases of Matter Protected by Time-Translation Symmetry," *Physical Review X*, vol. 7, no. 1, p. 011 026, Mar. 2017, Publisher: American Physical Society, DOI: 10.1103/PhysRevX.7.011026.

[59] D. Abanin, W. De Roeck, W. W. Ho, and F. Huveneers, "A Rigorous Theory of Many-Body Prethermalization for Periodically Driven and Closed Quantum Systems," en, *Communications in Mathematical Physics*, vol. 354, no. 3, pp. 809–827, Sep. 2017, ISSN: 1432-0916, DOI: 10.1007/s00220-017-2930-x.

[60] F. Machado, D. V. Else, G. D. Kahanamoku-Meyer, C. Nayak, and N. Y. Yao, "Long-Range Prethermal Phases of Nonequilibrium Matter," *Physical Review X*, vol. 10, no. 1, p. 011 043, Feb. 2020, DOI: 10.1103/PhysRevX.10.011043.

[61] F. Machado, G. D. Kahanamoku-Meyer, D. V. Else, C. Nayak, and N. Y. Yao, "Exponentially slow heating in short and long-range interacting Floquet systems," *Physical Review Research*, vol. 1, no. 3, p. 033 202, Dec. 2019, Publisher: American Physical Society, DOI: 10.1103/PhysRevResearch.1.033202.

[62] F. Pietracaprina, N. Macé, D. J. Luitz, and F. Alet, "Shift-invert diagonalization of large many-body localizing spin chains," en, *SciPost Physics*, vol. 5, no. 5, p. 045, Nov. 2018, ISSN: 2542-4653, DOI: 10.21468/SciPostPhys.5.5.045.

[63] D. J. Luitz, N. Laflorencie, and F. Alet, "Many-body localization edge in the random-field Heisenberg chain," *Physical Review B*, vol. 91, no. 8, p. 081 103, Feb. 2015, Publisher: American Physical Society, DOI: 10.1103/PhysRevB.91.081103.

[64] D. A. Abanin, E. Altman, I. Bloch, and M. Serbyn, "Colloquium: Many-body localization, thermalization, and entanglement," *Reviews of Modern Physics*, vol. 91, no. 2, p. 021 001, May 2019, Publisher: American Physical Society, DOI: 10.1103/RevModPhys.91.021001.

[65] A. Kitaev, *A simple model of quantum holography (part 1)*, Apr. 2015, [Online]. Available: https://online.kitp.ucsb.edu/online/entangled15/kitaev/ (visited on 04/29/2023).

[66] J. Maldacena and D. Stanford, "Remarks on the Sachdev-Ye-Kitaev model," *Physical Review D*, vol. 94, no. 10, p. 106 002, Nov. 2016, Publisher: American Physical Society, DOI: 10.1103/PhysRevD.94.106002.

[67] J. Maldacena, S. H. Shenker, and D. Stanford, "A bound on chaos," en, *Journal of High Energy Physics*, vol. 2016, no. 8, p. 106, Aug. 2016, ISSN: 1029-8479, DOI: 10.1007/JHEP08(2016)106.

[68] P. Gao, D. L. Jafferis, and A. C. Wall, "Traversable wormholes via a double trace deformation," en, *Journal of High Energy Physics*, vol. 2017, no. 12, p. 151, Dec. 2017, ISSN: 1029-8479, DOI: 10.1007/JHEP12(2017)151.

[69] J. Maldacena, D. Stanford, and Z. Yang, "Diving into traversable wormholes," en, *Fortschritte der Physik*, vol. 65, no. 5, p. 1 700 034, 2017, _eprint: https://onlinelibrary.wiley.com/doi ISSN: 1521-3978, DOI: 10.1002/prop.201700034.

[70] A. R. Brown, H. Gharibyan, S. Leichenauer, *et al.*, "Quantum Gravity in the Lab. I. Teleportation by Size and Traversable Wormholes," *PRX Quantum*, vol. 4, no. 1, p. 010 320, Feb. 2023, Publisher: American Physical Society, DOI: 10.1103/PRXQuantum.4.010320.

[71] T. Schuster, B. Kobrin, P. Gao, *et al.*, "Many-Body Quantum Teleportation via Operator Spreading in the Traversable Wormhole Protocol," *Physical Review X*, vol. 12, no. 3, p. 031 013, Jul. 2022, Publisher: American Physical Society, DOI: 10.1103/PhysRevX.12.031013.

[72] C. Bartsch and J. Gemmer, "Dynamical Typicality of Quantum Expectation Values," *Physical Review Letters*, vol. 102, no. 11, p. 110 403, Mar. 2009, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.102.110403.

[73] R. Steinigeweg, J. Gemmer, and W. Brenig, "Spin-Current Autocorrelations from Single Pure-State Propagation," *Physical Review Letters*, vol. 112, no. 12, p. 120 601, Mar. 2014, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.112.120601.

[74] B. Kobrin, Z. Yang, G. D. Kahanamoku-Meyer, *et al.*, "Many-Body Chaos in the Sachdev-Ye-Kitaev Model," *Physical Review Letters*, vol. 126, no. 3, p. 030 602, Jan. 2021, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.126.030602.

[75] E. Cáceres, A. Misobuchi, and R. Pimentel, "Sparse SYK and traversable wormholes," en, *Journal of High Energy Physics*, vol. 2021, no. 11, p. 15, Nov. 2021, ISSN: 1029-8479, DOI: 10.1007/JHEP11(2021)015.

[76] J. E. Roman, C. Campos, L. Dalcin, E. Romero, and A. Tomas, "SLEPc Users Manual," D. Sistemes Informàtics i Computació, Universitat Politècnica de València, Tech. Rep. DSIC-II/24/02 - Revision 3.19, 2023, [Online]. Available: https://slepc.upv.es/documentation/ (visited on 04/26/2023).

[77] V. Hernandez, J. E. Roman, A. Tomas, and V. Vidal, "Krylov-Schur Methods in SLEPc," Universitat Politècnica de València, Tech. Rep. STR-7, 2007.

[78] R. B. Sidje, "Expokit: A software package for computing matrix exponentials," *ACM Transactions on Mathematical Software*, vol. 24, no. 1, pp. 130–156, Mar. 1998, ISSN: 0098-3500, DOI: 10.1145/285861.285868.

[79] J. E. Roman, *[petsc-users] Solve for all repeated eigenvalues*, Jun. 2018, [Online]. Available: https://lists.mcs.anl.gov/pipermail/petsc-users/2018-June/035618.html (visited on 05/04/2023).

[80] T. J. Park and J. C. Light, "Unitary quantum time evolution by iterative Lanczos reduction," *The Journal of Chemical Physics*, vol. 85, no. 10, pp. 5870–5876, Nov. 1986, ISSN: 0021-9606, DOI: 10.1063/1.451548.

[81] Y. Saad, "Analysis of Some Krylov Subspace Approximations to the Matrix Exponential Operator," *SIAM Journal on Numerical Analysis*, vol. 29, no. 1, pp. 209–228, Feb. 1992, Publisher: Society for Industrial and Applied Mathematics, ISSN: 0036-1429, DOI: 10.1137/0729014.

[82] D. E. Stewart and T. S. Leyk, "Error estimates for Krylov subspace approximations of matrix exponentials," en, *Journal of Computational and Applied Mathematics*, vol. 72, no. 2, pp. 359–369, Aug. 1996, ISSN: 0377-0427, DOI: 10.1016/0377-0427(96)00006-4.

[83] M. Hochbruck and C. Lubich, "On Krylov Subspace Approximations to the Matrix Exponential Operator," *SIAM Journal on Numerical Analysis*, vol. 34, no. 5, pp. 1911–1925, Oct. 1997, Publisher: Society for Industrial and Applied Mathematics, ISSN: 0036-1429, DOI: 10.1137/S0036142995280572.

[84] C. Moler and C. Van Loan, "Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later," *SIAM Review*, vol. 45, no. 1, pp. 3–49, Jan. 2003, Publisher: Society for Industrial and Applied Mathematics, ISSN: 0036-1445, DOI: 10.1137/S00361445024180.

[85] T. Jawecki, W. Auzinger, and O. Koch, "Computable upper error bounds for Krylov approximations to matrix exponentials and associated $${\varvec{\varphi }}$$-functions," en, *BIT Numerical Mathematics*, vol. 60, no. 1, pp. 157–197, Mar. 2020, ISSN: 1572-9125, DOI: 10.1007/s10543-019-00771-6.

[86] J. M. Ruffinelli, E. M. Fortes, D. A. Wisniacki, and M. Larocca, "Loschmidt-echo approach to error estimation in Krylov-subspace approximation," *Physical Review A*, vol. 106, no. 4, p. 042 423, Oct. 2022, Publisher: American Physical Society, DOI: 10.1103/PhysRevA.106.042423.

[87] *Architecture - NERSC Documentation*, [Online]. Available: https://docs.nersc.gov/systems/perlmutter/architecture/ (visited on 06/02/2023).

[88] M. Hauru, A. Morningstar, J. Beall, M. Ganahl, A. Lewis, and G. Vidal, *Simulation of quantum physics with Tensor Processing Units: Brute-force computation of ground states and time evolution*, arXiv:2111.10466 [cond-mat, physics:quant-ph], Nov. 2021, DOI: 10.48550/arXiv.2111.10466.

[89] P. W. Anderson, "Absence of Diffusion in Certain Random Lattices," *Physical Review*, vol. 109, no. 5, pp. 1492–1505, Mar. 1958, Publisher: American Physical Society, DOI: 10.1103/PhysRev.109.1492.

[90] M. Schreiber, S. S. Hodgman, P. Bordia, *et al.*, "Observation of many-body localization of interacting fermions in a quasirandom optical lattice," *Science*, vol. 349, no. 6250, pp. 842–845, Aug. 2015, Publisher: American Association for the Advancement of Science, DOI: 10.1126/science.aaa7432.

[91] J.-y. Choi, S. Hild, J. Zeiher, *et al.*, "Exploring the many-body localization transition in two dimensions," *Science*, vol. 352, no. 6293, pp. 1547–1552, Jun. 2016, Publisher: American Association for the Advancement of Science, DOI: 10.1126/science.aaf8834.

[92] J. Smith, A. Lee, P. Richerme, *et al.*, "Many-body localization in a quantum simulator with programmable random disorder," en, *Nature Physics*, vol. 12, no. 10, pp. 907–911, Oct. 2016, Number: 10 Publisher: Nature Publishing Group, ISSN: 1745-2481, DOI: 10.1038/nphys3783.

[93] T. Kohlert, S. Scherg, X. Li, *et al.*, "Observation of Many-Body Localization in a One-Dimensional System with a Single-Particle Mobility Edge," *Physical Review Letters*, vol. 122, no. 17, p. 170 403, May 2019, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.122.170403.

[94] P. Bordia, H. Lüschen, S. Scherg, *et al.*, "Probing Slow Relaxation and Many-Body Localization in Two-Dimensional Quasiperiodic Systems," *Physical Review X*, vol. 7, no. 4, p. 041 047, Nov. 2017, Publisher: American Physical Society, DOI: 10.1103/PhysRevX.7.041047.

[95] A. Lukin, M. Rispoli, R. Schittko, *et al.*, "Probing entanglement in a many-body–localized system," *Science*, vol. 364, no. 6437, pp. 256–260, Apr. 2019, Publisher: American Association for the Advancement of Science, DOI: 10.1126/science.aau0818.

[96] S. Johri, R. Nandkishore, and R. N. Bhatt, "Many-Body Localization in Imperfectly Isolated Quantum Systems," *Physical Review Letters*, vol. 114, no. 11, p. 117 401, Mar. 2015, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.114.117401.

[97] E. Cuevas, M. Feigel'man, L. Ioffe, and M. Mezard, "Level statistics of disordered spin-1/2 systems and materials with localized Cooper pairs," en, *Nature Communications*, vol. 3, no. 1, p. 1128, Oct. 2012, Number: 1 Publisher: Nature Publishing Group, ISSN: 2041-1723, DOI: 10.1038/ncomms2115.

[98] B. Bauer and C. Nayak, "Area laws in a many-body localized state and its implications for topological order," en, *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2013, no. 09, P09005, Sep. 2013, Publisher: IOP Publishing and SISSA, ISSN: 1742-5468, DOI: 10.1088/1742-5468/2013/09/P09005.

[99] A. Pal and D. A. Huse, "Many-body localization phase transition," *Physical Review B*, vol. 82, no. 17, p. 174 411, Nov. 2010, Publisher: American Physical Society, DOI: 10.1103/PhysRevB.82.174411.

[100] W. De Roeck, F. Huveneers, M. Müller, and M. Schiulaz, "Absence of many-body mobility edges," *Physical Review B*, vol. 93, no. 1, p. 014 203, Jan. 2016, Publisher: American Physical Society, DOI: 10.1103/PhysRevB.93.014203.

[101] J. Šuntajs, J. Bonča, T. Prosen, and L. Vidmar, "Quantum chaos challenges many-body localization," *Physical Review E*, vol. 102, no. 6, p. 062 144, Dec. 2020, Publisher: American Physical Society, DOI: 10.1103/PhysRevE.102.062144.

[102] P. Ghysels, S. L. Xiaoye, C. Gorman, and F.-H. Rouet, "A Robust Parallel Preconditioner for Indefinite Systems Using Hierarchical Matrices and Randomized Sampling," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, ISSN: 1530-2075, May 2017, pp. 897–906, DOI: 10.1109/IPDPS.2017.21.

[103] V. Oganesyan and D. A. Huse, "Localization of interacting fermions at high temperature," *Physical Review B*, vol. 75, no. 15, p. 155 111, Apr. 2007, Publisher: American Physical Society, DOI: 10.1103/PhysRevB.75.155111.

[104] D. N. Page, "Average entropy of a subsystem," *Physical Review Letters*, vol. 71, no. 9, pp. 1291–1294, Aug. 1993, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.71.1291.

[105] A. V. Knyazev, "Toward the Optimal Preconditioned Eigensolver: Locally Optimal Block Preconditioned Conjugate Gradient Method," *SIAM Journal on Scientific Computing*, vol. 23, no. 2, pp. 517–541, Jan. 2001, Publisher: Society for Industrial and Applied Mathematics, ISSN: 1064-8275, DOI: 10.1137/S1064827500366124.

[106] J. A. Duersch, M. Shao, C. Yang, and M. Gu, "A Robust and Efficient Implementation of LOBPCG," *SIAM Journal on Scientific Computing*, vol. 40, no. 5, pp. C655–C676, Jan. 2018, Publisher: Society for Industrial and Applied Mathematics, ISSN: 1064-8275, DOI: 10.1137/17M1129830.

[107] L.-W. Wang and A. Zunger, "Solving Schrödinger's equation around a desired energy: Application to silicon quantum dots," *The Journal of Chemical Physics*, vol. 100, no. 3, pp. 2394–2397, Feb. 1994, Publisher: American Institute of Physics, ISSN: 0021-9606, DOI: 10.1063/1.466486.

[108] C. J. Jia, Y. Wang, C. B. Mendl, B. Moritz, and T. P. Devereaux, "Paradeisos: A perfect hashing algorithm for many-body eigenvalue problems," en, *Computer Physics Communications*, vol. 224, pp. 81–89, Mar. 2018, ISSN: 0010-4655, DOI: 10.1016/j.cpc.2017.11.011.

[109] M. Röhrig-Zöllner, J. Thies, M. Kreutzer, *et al.*, "Increasing the Performance of the Jacobi–Davidson Method by Blocking," *SIAM Journal on Scientific Computing*, vol. 37, no. 6, pp. C697–C722, Jan. 2015, Publisher: Society for Industrial and Applied Mathematics, ISSN: 1064-8275, DOI: 10.1137/140976017.

[110] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th national conference*, ser. ACM '69, New York, NY, USA: Association for Computing Machinery, Aug. 1969, pp. 157–172, ISBN: 978-1-4503-7493-4, DOI: 10.1145/800195.805928.

[111] E. Farhi and A. W. Harrow, *Quantum Supremacy through the Quantum Approximate Optimization Algorithm*, arXiv:1602.07674 [quant-ph], Oct. 2019, DOI: 10.48550/arXiv.1602.07674.

[112] S. Bravyi, D. Gosset, and R. König, "Quantum advantage with shallow circuits," en, *Science*, vol. 362, no. 6412, pp. 308–311, Oct. 2018, ISSN: 0036-8075, 1095-9203, DOI: 10.1126/science.aar3106.

[113] S. Bravyi, D. Gosset, R. König, and M. Tomamichel, "Quantum advantage with noisy shallow circuits," en, *Nature Physics*, vol. 16, no. 10, pp. 1040–1045, Oct. 2020, Number: 10 Publisher: Nature Publishing Group, ISSN: 1745-2481, DOI: 10.1038/s41567-020-0948-z.

[114] S. Aaronson and S. Gunn, "On the Classical Hardness of Spoofing Linear Cross-Entropy Benchmarking," *Theory of Computing*, vol. 16, pp. 1–8, Nov. 2020, Number: 11 Publisher: Theory of Computing, DOI: 10.4086/toc.2020.v016a011.

[115] G. D. Kahanamoku-Meyer, S. Choi, U. V. Vazirani, and N. Y. Yao, "Classically-Verifiable Quantum Advantage from a Computational Bell Test," *Nature Physics*, vol. 18, no. 8, pp. 918–924, Aug. 2022, arXiv:2104.00687 [quant-ph], ISSN: 1745-2473, 1745-2481, DOI: 10.1038/s41567-022-01643-7.

[116] D. Shepherd and M. J. Bremner, "Temporally unstructured quantum computation," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 465, no. 2105, pp. 1413–1439, May 2009, DOI: 10.1098/rspa.2008.0443.

[117] G. D. Kahanamoku-Meyer, *GregDMeyer/IQPwn: V1.0*, Jan. 2023, DOI: 10.5281/zenodo.7545882.

[118] Y. Alnawakhtha, A. Mantri, C. A. Miller, and D. Wang, *Lattice-Based Quantum Advantage from Rotated Measurements*, arXiv:2210.10143 [quant-ph], Oct. 2022, DOI: 10.48550/arXiv.2210.10143.

[119] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, Oct. 1997, ISSN: 0097-5397, DOI: 10.1137/S0097539795293172.

[120] D. Aharonov, M. Ben-Or, E. Eban, and U. Mahadev, "Interactive Proofs for Quantum Computations," *arXiv:1704.04487 [quant-ph]*, Apr. 2017, arXiv: 1704.04487, [Online]. Available: http://arxiv.org/abs/1704.04487 (visited on 03/16/2021).

[121] J. Watrous, "PSPACE Has Constant-Round Quantum Interactive Proof Systems," in *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, ser. FOCS '99, USA: IEEE Computer Society, Oct. 1999, p. 112, ISBN: 978-0-7695-0409-4, (visited on 06/23/2022).

[122] A. Kitaev and J. Watrous, "Parallelization, amplification, and exponential time simulation of quantum interactive proof systems," in *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, ser. STOC '00, New York, NY, USA: Association for Computing Machinery, May 2000, pp. 608–617, ISBN: 978-1-58113-184-0, DOI: 10.1145/335305.335387.

[123] H. Kobayashi and K. Matsumoto, "Quantum multi-prover interactive proof systems with limited prior entanglement," en, *Journal of Computer and System Sciences*, vol. 66, no. 3, pp. 429–450, May 2003, ISSN: 0022-0000, DOI: 10.1016/S0022-0000(03)00035-7.

[124] J. Fitzsimons and T. Vidick, "A Multiprover Interactive Proof System for the Local Hamiltonian Problem," in *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ser. ITCS '15, New York, NY, USA: Association for Computing Machinery, Jan. 2015, pp. 103–112, ISBN: 978-1-4503-3333-7, DOI: 10.1145/2688073.2688094.

[125] I. L. Markov, A. Fatima, S. V. Isakov, and S. Boixo, "Massively Parallel Approximate Simulation of Hard Quantum Circuits," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, ISSN: 0738-100X, Jul. 2020, pp. 1–6, DOI: 10.1109/DAC18072.2020.9218591.

[126] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM*, vol. 56, no. 6, 34:1–34:40, Sep. 2009, ISSN: 0004-5411, DOI: 10.1145/1568318.1568324.

[127] J. S. Bell, "On the Einstein Podolsky Rosen paradox," *Physics Physique Fizika*, vol. 1, no. 3, pp. 195–200, Nov. 1964, Publisher: American Physical Society, DOI: 10.1103/PhysicsPhysiqueFizika.1.195.

[128] J. F. Clauser, M. A. Horne, A. Shimony, and R. A. Holt, "Proposed Experiment to Test Local Hidden-Variable Theories," *Physical Review Letters*, vol. 23, no. 15, pp. 880–884, Oct. 1969, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.23.880.

[129] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976, Conference Name: IEEE Transactions on Information Theory, ISSN: 1557-9654, DOI: 10.1109/TIT.1976.1055638.

[130] C. Peikert and B. Waters, "Lossy trapdoor functions and their applications," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*, ser. STOC '08, New York, NY, USA: Association for Computing Machinery, May 2008, pp. 187–196, ISBN: 978-1-60558-047-0, DOI: 10.1145/1374376.1374406.

[131] D. M. Freeman, O. Goldreich, E. Kiltz, A. Rosen, and G. Segev, "More Constructions of Lossy and Correlation-Secure Trapdoor Functions," en, in *Public Key Cryptography – PKC 2010*, P. Q. Nguyen and D. Pointcheval, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2010, pp. 279–295, ISBN: 978-3-642-13013-7, DOI: 10.1007/978-3-642-13013-7_17.

[132] M. O. Rabin, "DIGITALIZED SIGNATURES AND PUBLIC-KEY FUNCTIONS AS INTRACTABLE AS FACTORIZATION," Massachusetts Institute of Technology, USA, Technical Report, 1979.

[133] S. Goldwasser, S. Micali, and R. L. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, Apr. 1988, Publisher: Society for Industrial and Applied Mathematics, ISSN: 0097-5397, DOI: 10.1137/0217017.

[134] V. S. Miller, "Use of Elliptic Curves in Cryptography," en, in *Advances in Cryptology — CRYPTO '85 Proceedings*, H. C. Williams, Ed., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 1986, pp. 417–426, ISBN: 978-3-540-39799-1, DOI: 10.1007/3-540-39799-X_31.

[135] N. Koblitz, "Elliptic curve cryptosystems," en, *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987, ISSN: 0025-5718, 1088-6842, DOI: 10.1090/S0025-5718-1987-0866109-5.

[136] E. Barker, "Recommendation for Key Management Part 1: General," en, National Institute of Standards and Technology, Tech. Rep. NIST SP 800-57pt1r4, Jan. 2016, NIST SP 800–57pt1r4, DOI: 10.6028/NIST.SP.800-57pt1r4.

[137] C. H. Bennett, "Time/Space Trade-Offs for Reversible Computation," *SIAM Journal on Computing*, vol. 18, no. 4, pp. 766–776, Aug. 1989, Publisher: Society for Industrial and Applied Mathematics, ISSN: 0097-5397, DOI: 10.1137/0218053.

[138] R. Y. Levine and A. T. Sherman, "A Note on Bennett's Time-Space Tradeoff for Reversible Computation," *SIAM Journal on Computing*, vol. 19, no. 4, pp. 673–677, Aug. 1990, Publisher: Society for Industrial and Applied Mathematics, ISSN: 0097-5397, DOI: 10.1137/0219046.

[139] D. Aharonov, A. Kitaev, and N. Nisan, "Quantum circuits with mixed states," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, ser. STOC '98, New York, NY, USA: Association for Computing Machinery, May 1998, pp. 20–30, ISBN: 978-0-89791-962-3, DOI: 10.1145/276698.276708.

[140] H. Babu, M. Islam, S. Chowdhury, and A. Chowdhury, "Synthesis of full-adder circuit using reversible logic," in *17th International Conference on VLSI Design. Proceedings.*, Jan. 2004, pp. 757–760, DOI: 10.1109/ICVD.2004.1261020.

[141] S. Kotiyal, H. Thapliyal, and N. Ranganathan, "Circuit for Reversible Quantum Multiplier Based on Binary Tree Optimizing Ancilla and Garbage Bits," in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, ISSN: 2380-6923, Jan. 2014, pp. 545–550, DOI: 10.1109/VLSID.2014.101.

[142] M. Saffman, "Quantum computing with atomic qubits and Rydberg interactions: Progress and challenges," *Journal of Physics B: Atomic, Molecular and Optical Physics*, vol. 49, no. 20, p. 202 001, 2016, Publisher: IOP Publishing.

[143] H. Levine, A. Keesling, G. Semeghini, *et al.*, "Parallel Implementation of High-Fidelity Multiqubit Gates with Neutral Atoms," *Physical Review Letters*, vol. 123, no. 17, p. 170 503, Oct. 2019, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.123.170503.

[144] T. Graham, M. Kwon, B. Grinkemeyer, *et al.*, "Rydberg-mediated entanglement in a two-dimensional neutral atom qubit array," *Physical Review Letters*, vol. 123, no. 23, p. 230 501, 2019, Publisher: APS.

[145] I. S. Madjarov, J. P. Covey, A. L. Shaw, *et al.*, "High-fidelity entanglement and detection of alkaline-earth Rydberg atoms," en, *Nature Physics*, vol. 16, no. 8, pp. 857–861, Aug. 2020, Number: 8 Publisher: Nature Publishing Group, ISSN: 1745-2481, DOI: 10.1038/s41567-020-0903-z.

[146] A. Browaeys and T. Lahaye, "Many-body physics with individually controlled Rydberg atoms," en, *Nature Physics*, vol. 16, no. 2, pp. 132–142, Feb. 2020, Number: 2 Publisher: Nature Publishing Group, ISSN: 1745-2481, DOI: 10.1038/s41567-019-0733-z.

[147] Z. Liu and A. Gheorghiu, "Depth-efficient proofs of quantumness," en-GB, *Quantum*, vol. 6, p. 807, Sep. 2022, Publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften, DOI: 10.22331/q-2022-09-19-807.

[148] S. Hirahara and F. Le Gall, "Test of Quantumness with Small-Depth Quantum Circuits," in *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, F. Bonchi and S. J. Puglisi, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), ISSN: 1868-8969, vol. 202, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 59:1–59:15, ISBN: 978-3-95977-201-3, DOI: 10.4230/LIPIcs.MFCS.2021.59.

[149] O. Goldreich and L. A. Levint, "A hard-core predicate for all one-way functions," in *In Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, 1989, pp. 25–32.

[150] D. Zhu, G. D. Kahanamoku-Meyer, L. Lewis, *et al.*, "Interactive Protocols for Classically-Verifiable Quantum Advantage," *arXiv:2112.05156 [cond-mat, physics:quant-ph]*, Dec. 2021, arXiv: 2112.05156, [Online]. Available: http://arxiv.org/abs/2112.05156 (visited on 01/26/2022).

[151] C. Ryan-Anderson, J. G. Bohnet, K. Lee, *et al.*, "Realization of Real-Time Fault-Tolerant Quantum Error Correction," *Physical Review X*, vol. 11, no. 4, p. 041 058, Dec. 2021, Publisher: American Physical Society, DOI: 10.1103/PhysRevX.11.041058.

[152] K. Pietrzak, "Cryptography from Learning Parity with Noise," en, in *SOFSEM 2012: Theory and Practice of Computer Science*, M. Bieliková, G. Friedrich, G. Gottlob, S. Katzenbeisser, and G. Turán, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2012, pp. 99–114, ISBN: 978-3-642-27660-6, DOI: 10.1007/978-3-642-27660-6_9.

[153] D. R. Simon, "On the Power of Quantum Computation," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1474–1483, Oct. 1997, Publisher: Society for Industrial and Applied Mathematics, ISSN: 0097-5397, DOI: 10.1137/S0097539796298637.

[154] T. Häner, S. Jaques, M. Naehrig, M. Roetteler, and M. Soeken, "Improved Quantum Circuits for Elliptic Curve Discrete Logarithms," en, in *Post-Quantum Cryptography*, J. Ding and J.-P. Tillich, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2020, pp. 425–444, ISBN: 978-3-030-44223-1, DOI: 10.1007/978-3-030-44223-1_23.

[155] G. Meyer, *GregDMeyer/quantum-advantage: V1.1*, May 2022, DOI: 10.5281/zenodo.6519250.

[156] T. G. Draper, "Addition on a Quantum Computer," *arXiv:quant-ph/0008033*, Aug. 2000, arXiv: quant-ph/0008033, [Online]. Available: http://arxiv.org/abs/quant-ph/0008033 (visited on 01/24/2020).

[157] S. Beauregard, "Circuit for Shor's algorithm using 2n+3 qubits," *Quantum Information & Computation*, vol. 3, no. 2, pp. 175–185, Mar. 2003, ISSN: 1533-7146.

[158] J. Zhang, G. Pagano, P. W. Hess, *et al.*, "Observation of a many-body dynamical phase transition with a 53-qubit quantum simulator," *Nature*, vol. 551, no. 7682, pp. 601–604, 2017, Publisher: Nature Publishing Group.

[159] P. Scholl, M. Schuler, H. J. Williams, *et al.*, "Quantum simulation of 2D antiferromagnets with hundreds of Rydberg atoms," en, *Nature*, vol. 595, no. 7866, pp. 233–238, Jul. 2021, Number: 7866 Publisher: Nature Publishing Group, ISSN: 1476-4687, DOI: 10.1038/s41586-021-03585-1.

[160] S. Ebadi, T. T. Wang, H. Levine, *et al.*, "Quantum phases of matter on a 256-atom programmable quantum simulator," en, *Nature*, vol. 595, no. 7866, pp. 227–232, Jul. 2021, Number: 7866 Publisher: Nature Publishing Group, ISSN: 1476-4687, DOI: 10.1038/s41586-021-03582-4.

[161] Y. Wang, X. Zhang, T. A. Corcovilos, A. Kumar, and D. S. Weiss, "Coherent Addressing of Individual Neutral Atoms in a 3D Optical Lattice," *Physical Review Letters*, vol. 115, no. 4, p. 043 003, Jul. 2015, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.115.043003.

[162] Y. Wang, A. Kumar, T.-Y. Wu, and D. S. Weiss, "Single-qubit gates based on targeted phase shifts in a 3D neutral atom array," *Science*, vol. 352, no. 6293, pp. 1562–1565, Jun. 2016, Publisher: American Association for the Advancement of Science, DOI: 10.1126/science.aaf2581.

[163] A. Kumar, T.-Y. Wu, F. Giraldo, and D. S. Weiss, "Sorting ultracold atoms in a three-dimensional optical lattice in a realization of Maxwell's demon," en, *Nature*, vol. 561, no. 7721, pp. 83–87, Sep. 2018, Number: 7721 Publisher: Nature Publishing Group, ISSN: 1476-4687, DOI: 10.1038/s41586-018-0458-7.

[164] R. Löw, H. Weimer, J. Nipper, *et al.*, "An experimental and theoretical guide to strongly interacting Rydberg gases," en, *Journal of Physics B: Atomic, Molecular and Optical Physics*, vol. 45, no. 11, p. 113 001, May 2012, Publisher: IOP Publishing, ISSN: 0953-4075, DOI: 10.1088/0953-4075/45/11/113001.

[165] S. de Léséleuc, D. Barredo, V. Lienhard, A. Browaeys, and T. Lahaye, "Analysis of imperfections in the coherent optical excitation of single atoms to Rydberg states," *Physical Review A*, vol. 97, no. 5, p. 053 803, May 2018, Publisher: American Physical Society, DOI: 10.1103/PhysRevA.97.053803.

[166] Y. Liu, Y. Sun, Z. Fu, *et al.*, "Infidelity Induced by Ground-Rydberg Decoherence of the Control Qubit in a Two-Qubit Rydberg-Blockade Gate," *Physical Review Applied*, vol. 15, no. 5, p. 054 020, May 2021, Publisher: American Physical Society, DOI: 10.1103/PhysRevApplied.15.054020.

[167] V. M. Schäfer, C. J. Ballance, K. Thirumalai, *et al.*, "Fast quantum logic gates with trapped-ion qubits," en, *Nature*, vol. 555, no. 7694, pp. 75–78, Mar. 2018, Number: 7694 Publisher: Nature Publishing Group, ISSN: 1476-4687, DOI: 10.1038/nature25737.

[168] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, 10th anniversary ed. USA: Cambridge University Press, 2011, ISBN: 978-1-107-00217-3.

[169] V. V. Shende and I. L. Markov, "On the CNOT-cost of TOFFOLI gates," *Quantum Information & Computation*, vol. 9, no. 5, pp. 461–486, May 2009, ISSN: 1533-7146.

[170] A. Barenco, C. H. Bennett, R. Cleve, *et al.*, "Elementary gates for quantum computation," *Physical Review A*, vol. 52, no. 5, pp. 3457–3467, Nov. 1995, Publisher: American Physical Society, DOI: 10.1103/PhysRevA.52.3457.

[171] S. Puri, L. St-Jean, J. A. Gross, *et al.*, "Bias-preserving gates with stabilized cat qubits," en, *Science Advances*, vol. 6, no. 34, eaay5901, Aug. 2020, Publisher: American Association for the Advancement of Science Section: Research Article, ISSN: 2375-2548, DOI: 10.1126/sciadv.aay5901.

[172] J. M. Pollard, "Theorems on factorization and primality testing," en, *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 76, no. 3, pp. 521–528, Nov. 1974, Publisher: Cambridge University Press, ISSN: 1469-8064, 0305-0041, DOI: 10.1017/S0305004100049252.

[173] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978, ISSN: 0001-0782, DOI: 10.1145/359340.359342.

[174] *CADO-NFS*, [Online]. Available: http://cado-nfs.gforge.inria.fr/ (visited on 06/27/2020).

[175] p. zimmermann paul, *[Cado-nfs-discuss] Factorization of RSA-250*, Library Catalog: Mailman, Feb. 2020, [Online]. Available: https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2020-February/001166.html (visited on 06/27/2020).

[176] C. Gidney, "Asymptotically Efficient Quantum Karatsuba Multiplication," *arXiv:1904.07356 [quant-ph]*, Apr. 2019, arXiv: 1904.07356, [Online]. Available: http://arxiv.org/abs/1904.07356 (visited on 10/07/2020).

[177] C. Gidney and M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits," en-GB, *Quantum*, vol. 5, p. 433, Apr. 2021, Publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften, DOI: 10.22331/q-2021-04-15-433.

[178] V. Lyubashevsky, C. Peikert, and O. Regev, "On Ideal Lattices and Learning with Errors over Rings," *Journal of the ACM*, vol. 60, no. 6, 43:1–43:35, Nov. 2013, ISSN: 0004-5411, DOI: 10.1145/2535925.

[179] R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "Efficient software implementation of ring-LWE encryption," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15, San Jose, CA, USA: EDA Consortium, Mar. 2015, pp. 339–344, ISBN: 978-3-9815370-4-8, (visited on 11/12/2020).

[180] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact Ring-LWE Cryptoprocessor," en, in *Cryptographic Hardware and Embedded Systems – CHES 2014*, L. Batina and M. Robshaw, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2014, pp. 371–391, ISBN: 978-3-662-44709-3, DOI: 10.1007/978-3-662-44709-3_21.

[181] C. Zalka, "Fast versions of Shor's quantum factoring algorithm," *arXiv:quant-ph/9806084*, Jun. 1998, arXiv: quant-ph/9806084, [Online]. Available: http://arxiv.org/abs/quant-ph/9806084 (visited on 03/15/2021).

[182] A. Schönhage and V. Strassen, "Schnelle Multiplikation großer Zahlen," de, *Computing*, vol. 7, no. 3, pp. 281–292, Sep. 1971, ISSN: 1436-5057, DOI: 10.1007/BF02242355.

[183] L. Kowada, R. Portugal, and C. Figueiredo, "Reversible Karatsuba's Algorithm.," *J. UCS*, vol. 12, pp. 499–511, Jan. 2006.

[184] A. Parent, M. Roetteler, and M. Mosca, "Improved reversible and quantum circuits for Karatsuba-based integer multiplication," in *12th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2017)*, M. M. Wilde, Ed., ser. Leibniz International Proceedings in Informatics (LIPIcs), ISSN: 1868-8969, vol. 73, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 7:1–7:15, ISBN: 978-3-95977-034-7, DOI: 10.4230/LIPIcs.TQC.2017.7.

[185] P. L. Montgomery, "Modular multiplication without trial division," en, *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985, ISSN: 0025-5718, 1088-6842, DOI: 10.1090/S0025-5718-1985-0777282-X.

[186] K. Javeed, D. Irwin, and X. Wang, "Design and Performance Comparison of Modular Multipliers Implemented on FPGA Platform," en, in *Cloud Computing and Security*, X. Sun, A. Liu, H.-C. Chao, and E. Bertino, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2016, pp. 251–260, ISBN: 978-3-319-48671-0, DOI: 10.1007/978-3-319-48671-0_23.

[187] M. Morales-Sandoval and A. Diaz-Perez, "Scalable GF(p) Montgomery multiplier based on a digit–digit computation approach," en, *IET Computers & Digital Techniques*, vol. 10, no. 3, pp. 102–109, 2016, ISSN: 1751-861X, DOI: https://doi.org/10.1049/iet-cdt.2015.0055.

[188] Y. Yang, C. Wu, Z. Li, and J. Yang, "Efficient FPGA implementation of modular multiplication based on Montgomery algorithm," en, *Microprocessors and Microsystems*, vol. 47, pp. 209–215, Nov. 2016, ISSN: 0141-9331, DOI: 10.1016/j.micpro.2016.07.008.

[189] I. Cong, S. Choi, and M. D. Lukin, "Quantum convolutional neural networks," en, *Nature Physics*, vol. 15, no. 12, pp. 1273–1278, Dec. 2019, Number: 12 Publisher: Nature Publishing Group, ISSN: 1745-2481, DOI: 10.1038/s41567-019-0648-8.

[190] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, Feb. 1989, Publisher: Society for Industrial and Applied Mathematics, ISSN: 0097-5397, DOI: 10.1137/0218012.

[191] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," in *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, Oct. 1990, 2–10 vol.1, DOI: 10.1109/FSCS.1990.89518.

[192] A. Shamir, "IP = PSPACE," *Journal of the ACM*, vol. 39, no. 4, pp. 869–877, Oct. 1992, ISSN: 0004-5411, DOI: 10.1145/146585.146609.

[193] B. Hensen, H. Bernien, A. E. Dréau, *et al.*, "Loophole-free Bell inequality violation using electron spins separated by 1.3 kilometres," en, *Nature*, vol. 526, no. 7575, pp. 682–686, Oct. 2015, Number: 7575 Publisher: Nature Publishing Group, ISSN: 1476-4687, DOI: 10.1038/nature15759.

[194] L. K. Shalm, E. Meyer-Scott, B. G. Christensen, *et al.*, "Strong Loophole-Free Test of Local Realism," *Physical Review Letters*, vol. 115, no. 25, p. 250 402, Dec. 2015, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.115.250402.

[195] M. Giustina, M. A. M. Versteegh, S. Wengerowsky, *et al.*, "Significant-Loophole-Free Test of Bell's Theorem with Entangled Photons," *Physical Review Letters*, vol. 115, no. 25, p. 250 401, Dec. 2015, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.115.250401.

[196] W. K. Hensinger, "Quantum computer based on shuttling trapped ions," en, *Nature*, vol. 592, no. 7853, pp. 190–191, Apr. 2021, Bandiera_abtest: a Cg_type: News And Views Number: 7853 Publisher: Nature Publishing Group Subject_term: Quantum information, Quantum physics, Computer science, DOI: 10.1038/d41586-021-00844-z.

[197] D. Bluvstein, H. Levine, G. Semeghini, *et al.*, "A quantum processor based on coherent transport of entangled atom arrays," en, *Nature*, vol. 604, no. 7906, pp. 451–456, Apr. 2022, Number: 7906 Publisher: Nature Publishing Group, ISSN: 1476-4687, DOI: 10.1038/s41586-022-04592-6.

[198] J. M. Pino, J. M. Dreiling, C. Figgatt, *et al.*, "Demonstration of the trapped-ion quantum CCD computer architecture," en, *Nature*, vol. 592, no. 7853, pp. 209–213, Apr. 2021, Number: 7853 Publisher: Nature Publishing Group, ISSN: 1476-4687, DOI: 10.1038/s41586-021-03318-4.

[199] D. Kielpinski, C. Monroe, and D. J. Wineland, "Architecture for a large-scale ion-trap quantum computer," en, *Nature*, vol. 417, no. 6890, pp. 709–711, Jun. 2002, Number: 6890 Publisher: Nature Publishing Group, ISSN: 1476-4687, DOI: 10.1038/nature00784.

[200] Y. Wan, D. Kienzler, S. D. Erickson, *et al.*, "Quantum gate teleportation between separated qubits in a trapped-ion processor," *Science*, vol. 364, no. 6443, pp. 875–878, May 2019, Publisher: American Association for the Advancement of Science, DOI: 10.1126/science.aaw9415.

[201] O. Regev, "The Learning with Errors Problem (Invited Survey)," in *2010 IEEE 25th Annual Conference on Computational Complexity*, ISSN: 1093-0159, Jun. 2010, pp. 191–204, DOI: 10.1109/CCC.2010.26.

[202] S. Goldwasser, S. Micali, and R. Rivest, "A "Paradoxical" Solution To The Signature Problem," in *25th Annual Symposium on Foundations of Computer Science, 1984.*, ISSN: 0272-5428, Oct. 1984, pp. 441–448, DOI: 10.1109/SFCS.1984.715946.

[203] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom functions and lattices," in *Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT'12, Berlin, Heidelberg: Springer-Verlag, Apr. 2012, pp. 719–737, ISBN: 978-3-642-29010-7, DOI: 10.1007/978-3-642-29011-4_42.

[204] J. Alwen, S. Krenn, K. Pietrzak, and D. Wichs, "Learning with Rounding, Revisited," en, in *Advances in Cryptology – CRYPTO 2013*, vol. 8042, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 57–74, ISBN: 978-3-642-40041-4, DOI: 10.1007/978-3-642-40041-4_4.

[205] C. Monroe, W. C. Campbell, L.-M. Duan, *et al.*, "Programmable quantum simulations of spin systems with trapped ions," *Reviews of Modern Physics*, vol. 93, no. 2, p. 025 001, Apr. 2021, Publisher: American Physical Society, DOI: 10.1103/RevModPhys.93.025001.

[206] L. Egan, D. M. Debroy, C. Noel, *et al.*, "Fault-tolerant control of an error-corrected qubit," en, *Nature*, vol. 598, no. 7880, pp. 281–286, Oct. 2021, Number: 7880 Publisher: Nature Publishing Group, ISSN: 1476-4687, DOI: 10.1038/s41586-021-03928-y.

[207] A. D. Córcoles, M. Takita, K. Inoue, *et al.*, "Exploiting Dynamic Quantum Circuits in a Quantum Algorithm with Superconducting Qubits," *Physical Review Letters*, vol. 127, no. 10, p. 100 501, Aug. 2021, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.127.100501.

[208] K. Rudinger, G. J. Ribeill, L. C. Govia, *et al.*, "Characterizing Midcircuit Measurements on a Superconducting Qubit Using Gate Set Tomography," *Physical Review Applied*, vol. 17, no. 1, p. 014 014, Jan. 2022, Publisher: American Physical Society, DOI: 10.1103/PhysRevApplied.17.014014.

[209] B. Skinner, J. Ruhman, and A. Nahum, "Measurement-Induced Phase Transitions in the Dynamics of Entanglement," *Physical Review X*, vol. 9, no. 3, p. 031 009, Jul. 2019, Publisher: American Physical Society, DOI: 10.1103/PhysRevX.9.031009.

[210] Y. Li, X. Chen, and M. P. A. Fisher, "Quantum Zeno effect and the many-body entanglement transition," *Physical Review B*, vol. 98, no. 20, p. 205 136, Nov. 2018, Publisher: American Physical Society, DOI: 10.1103/PhysRevB.98.205136.

[211] C. Noel, P. Niroula, D. Zhu, *et al.*, "Measurement-induced quantum phases realized in a trapped-ion quantum computer," en, *Nature Physics*, vol. 18, no. 7, pp. 760–764, Jul. 2022, Number: 7 Publisher: Nature Publishing Group, ISSN: 1745-2481, DOI: 10.1038/s41567-022-01619-7.

[212] M. Cetina, L. Egan, C. Noel, *et al.*, "Control of Transverse Motion for Quantum Gates on Individually Addressed Atomic Qubits," *PRX Quantum*, vol. 3, no. 1, p. 010 334, Mar. 2022, Publisher: American Physical Society, DOI: 10.1103/PRXQuantum.3.010334.

[213] S. Olmschenk, K. C. Younge, D. L. Moehring, D. N. Matsukevich, P. Maunz, and C. Monroe, "Manipulation and detection of a trapped Yb^+ hyperfine qubit," *Physical Review A*, vol. 76, no. 5, p. 052 314, Nov. 2007, Publisher: American Physical Society, DOI: 10.1103/PhysRevA.76.052314.

[214] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe, "Demonstration of a small programmable quantum computer with atomic qubits," en, *Nature*, vol. 536, no. 7614, pp. 63–66, Aug. 2016, Number: 7614 Publisher: Nature Publishing Group, ISSN: 1476-4687, DOI: 10.1038/nature18648.

[215] K. Mølmer and A. Sørensen, "Multiparticle Entanglement of Hot Trapped Ions," *Physical Review Letters*, vol. 82, no. 9, pp. 1835–1838, Mar. 1999, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.82.1835.

[216] E. Solano, R. L. de Matos Filho, and N. Zagury, "Deterministic Bell states and measurement of the motional state of two trapped ions," *Physical Review A*, vol. 59, no. 4, R2539–R2543, Apr. 1999, Publisher: American Physical Society, DOI: 10.1103/PhysRevA.59.R2539.

[217] G. J. Milburn, S. Schneider, and D. F. V. James, "Ion Trap Quantum Computing with Warm Ions," en, in *Scalable Quantum Computers*, John Wiley & Sons, Ltd, 2000, pp. 31–40, ISBN: 978-3-527-60318-3, [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/3527603182.ch3 (visited on 05/24/2023).

[218] T. Choi, S. Debnath, T. A. Manning, *et al.*, "Optimal Quantum Control of Multimode Couplings between Trapped Ion Qubits for Scalable Entanglement," *Physical Review Letters*, vol. 112, no. 19, p. 190 502, May 2014, Publisher: American Physical Society, DOI: 10.1103/PhysRevLett.112.190502.

[219] P. Maunz, "High Optical Access Trap 2.0.," Report Number: SAND–2016-0796R, 1237003, 618951, Jan. 2016, SAND–2016–0796R, 1237003, 618951, DOI: 10.2172/1237003.

[220] F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé, and P. Zimmermann, "Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment," Tech. Rep. 697, 2020, [Online]. Available: https://eprint.iacr.org/2020/697 (visited on 12/09/2021).

[221] R. Lindner and C. Peikert, "Better Key Sizes (and Attacks) for LWE-Based Encryption," en, in *Topics in Cryptology – CT-RSA 2011*, A. Kiayias, Ed., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2011, pp. 319–339, ISBN: 978-3-642-19074-2, DOI: 10.1007/978-3-642-19074-2_21.

[222] S. Dutta, D. Bhattacharjee, and A. Chattopadhyay, "Quantum circuits for Toom-Cook multiplication," *Physical Review A*, vol. 98, no. 1, p. 012 311, Jul. 2018, Publisher: American Physical Society, DOI: 10.1103/PhysRevA.98.012311.

[223] H. T. Larasati, A. M. Awaludin, J. Ji, and H. Kim, "Quantum Circuit Design of Toom 3-Way Multiplication," en, *Applied Sciences*, vol. 11, no. 9, p. 3752, Jan. 2021, Number: 9 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2076-3417, DOI: 10.3390/app11093752.

[224] D. Beckman, A. N. Chari, S. Devabhaktuni, and J. Preskill, "Efficient networks for quantum factoring," *Physical Review A*, vol. 54, no. 2, pp. 1034–1063, Aug. 1996, DOI: 10.1103/PhysRevA.54.1034.

[225] D. E. Knuth, *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998, ISBN: 0-201-89684-2.

[226] M. Bodrato, "Towards Optimal Toom-Cook Multiplication for Univariate and Multivariate Polynomials in Characteristic 2 and 0," en, in *Arithmetic of Finite Fields*, C. Carlet and B. Sunar, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2007, pp. 116–133, ISBN: 978-3-540-73074-3, DOI: 10.1007/978-3-540-73074-3_10.

[227] *Multiplication Algorithms (GNU MP 6.2.1)*, [Online]. Available: https://gmplib.org/manual/Multiplication-Algorithms (visited on 06/04/2023).

[228] V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," *Physical Review A*, vol. 54, no. 1, pp. 147–153, Jul. 1996, Publisher: American Physical Society, DOI: 10.1103/PhysRevA.54.147.

[229] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," *arXiv:quant-ph/0410184*, Oct. 2004, [Online]. Available: http://arxiv.org/abs/quant-ph/0410184.

[230] T. G. Draper, S. A. Kutin, E. M. Rains, and K. M. Svore, "A logarithmic-depth quantum carry-lookahead adder," *Quantum Information & Computation*, vol. 6, no. 4, pp. 351–369, Jul. 2006, ISSN: 1533-7146.

[231] T. Häner, M. Roetteler, and K. M. Svore, "Factoring using 2n+2 qubits with Toffoli based modular multiplication," *arXiv:1611.07995 [quant-ph]*, Jun. 2017, arXiv: 1611.07995, [Online]. Available: `http://arxiv.org/abs/1611.07995` (visited on 11/05/2020).

[232] C. Gidney, "Factoring with n+2 clean qubits and n-1 dirty qubits," *arXiv:1706.07884 [quant-ph]*, Jan. 2018, arXiv: 1706.07884, [Online]. Available: `http://arxiv.org/abs/1706.07884` (visited on 06/25/2020).

[233] R. Cleve and J. Watrous, "Fast parallel circuits for the quantum Fourier transform," in *Proceedings 41st Annual Symposium on Foundations of Computer Science*, ISSN: 0272-5428, Nov. 2000, pp. 526–536, DOI: `10.1109/SFCS.2000.892140`.

[234] C. Gidney, *Windowed quantum arithmetic*, arXiv:1905.07682 [quant-ph], May 2019, DOI: `10.48550/arXiv.1905.07682`.

[235] L. Lewis, D. Zhu, A. Gheorghiu, *et al.*, *Experimental Implementation of an Efficient Test of Quantumness*, arXiv:2209.14316 [cond-mat, physics:quant-ph], Sep. 2022, DOI: `10.48550/arXiv.2209.14316`.

[236] H. Lenstra and A. Lenstra, *The development of the number field sieve* (Lecture Notes in Mathematics), eng. Berlin, Heidelberg: Springer, 1993, vol. 1554, Backup Publisher: SpringerLink (Online service) ISSN: 0075-8434, ISBN: 978-3-540-57013-4.

[237] U. Meyer, *Reconstruction of D. H. Lehmer's number theory computer with LEGO*, [Online]. Available: `https://brickshelf.com/gallery/ulimy/Computer/lehmer.pdf` (visited on 06/23/2023).

[238] M. R. Williams, *Lehmer Sieves*, [Online]. Available: `https://www.ed-thelen.org/comp-hist/Mike-Williams-Lehmer.html` (visited on 06/03/2023).

[239] A. Morningstar, M. Hauru, J. Beall, *et al.*, "Simulation of Quantum Many-Body Dynamics with Tensor Processing Units: Floquet Prethermalization," *PRX Quantum*, vol. 3, no. 2, p. 020 331, May 2022, Publisher: American Physical Society, DOI: `10.1103/PRXQuantum.3.020331`.

[240] S. Lee, J. Lee, H. Zhai, *et al.*, "Evaluating the evidence for exponential quantum advantage in ground-state quantum chemistry," en, *Nature Communications*, vol. 14, no. 1, p. 1952, Apr. 2023, Number: 1 Publisher: Nature Publishing Group, ISSN: 2041-1723, DOI: `10.1038/s41467-023-37587-6`.