On the practicality of quantum sieving algorithms for the shortest vector problem

Joao F. Doriguello*[1], George Giapitzakis[2], Alessandro Luongo[3], and Aditya Morolia[3]

[1]HUN-REN Alfréd Rényi Institute of Mathematics, Budapest, Hungary
[2]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
[3]Centre for Quantum Technologies, National University of Singapore, Singapore

October 17, 2024

Abstract

One of the main candidates of post-quantum cryptography is lattice-based cryptography. Its cryptographic security against quantum attackers is based on the worst-case hardness of lattice problems like the shortest vector problem (SVP), which asks to find the shortest non-zero vector in an integer lattice. Asymptotic quantum speedups for solving SVP are known and rely on Grover's search. However, to assess the security of lattice-based cryptography against these Grover-like quantum speedups, it is necessary to carry out a precise resource estimation beyond asymptotic scalings. In this work, we perform a careful analysis on the resources required to implement several sieving algorithms aided by Grover's search for dimensions of cryptographic interests. For such, we take into account fixed-point quantum arithmetic operations, non-asymptotic Grover's search, the cost of using quantum random access memory (QRAM), different physical architectures, and quantum error correction. We find that even under very optimistic assumptions like circuit-level noise of 10^{-5}, code cycles of 100 ns, reaction time of 1 μs, and using state-of-the-art arithmetic circuits and quantum error-correction protocols, the best sieving algorithms require $\approx 10^{13}$ physical qubits and $\approx 10^{31}$ years to solve SVP on a lattice of dimension 400, which is roughly the dimension for minimally secure post-quantum cryptographic standards currently being proposed by NIST. We estimate that a 6-GHz-clock-rate single-core classical computer would take roughly the same amount of time to solve the same problem. We conclude that there is currently little to no quantum speedup in the dimensions of cryptographic interest and the possibility of realising a considerable quantum speedup using quantum sieving algorithms would require significant breakthroughs in theoretical protocols and hardware development.

Contents

*Corresponding author: doriguello@renyi.hu

1 Introduction

Lattice-based cryptography [101, 174, 175, 153] has emerged as an important alternative to traditional discrete-log-based cryptosystems like RSA, DSA, and Elliptic-curve cryptography since the advent of Shor's algorithm in 1994 [186, 185]. Apart from the belief of being cryptographically secure against quantum attacks, lattice-based cryptography has several other important properties, like being based on *worst-case hardness* of lattice problems, e.g., the shortest vector problem (SVP) [8], and allowing fully homomorphic encryption schemes [82, 45]. For these reasons, lattice-based cryptography is still considered one of the main candidates of *post-quantum cryptography* [38], to the point of being one of the finalist in NIST's undertaking of the standardization of post-quantum cryptography schemes [162]. It is therefore of paramount importance to understand the security level provided by lattice-based cryptography not only against classical attackers but also against quantum ones in order to determine the security guaranteed at various parameter regimes.

The security assumptions of such schemes are related to the problem of finding the shortest non-zero vector in a lattice, in the sense that the best attacks on them make use of an oracle for (approximate) SVP. There are currently three main types of algorithms to solve SVP: sieving [12, 11, 156, 5], enumeration [76, 112, 168], and constructing the Voronoi cell of the lattice [6, 155]. Heuristic versions of lattice sieving and enumeration have seen a lot of success in solving SVP practically, with lattice sieving [119] holding the record for breaking an NTRU [101] challenge by Security Innovation Inc. [103] with largest dimension. By using the BKZ (Block-Korkine-Zolotarev) algorithm [183] with lattice sieving, Kirshanova, May, and Nowakowski [119] recently broke a lattice-based construction in dimension $D = 181$ in 20 core years. Despite this and a long line of work on such algorithms [112, 168, 76, 160, 156, 128], however, enumeration and sieving algorithms remain notoriously hard to analyze. The situation is further complicated by the introduction of quantum subroutines into sieving and enumeration algorithms like Grover's search [95, 96], which makes unclear how secure lattice-based cryptography is against these "quantumly-enhanced" algorithms. It is thus of critical importance to assess the actual quantum advantage that subroutines like Grover's search provide in solving SVP.

A few different works have tried to estimate the amount of resources required, and thus the computational advantage provided, by Grover's search in sieving [14, 116, 117] and in enumeration [28, 39, 171] algorithms. However, all of the existing work on such algorithms ignores the spacetime cost of quantum random access memory (QRAM) [89, 90] and/or of quantum error correction on fault tolerant quantum computers, which can add a significant overhead. In this work, we perform a very thorough analysis of the quantum resources required to enhance several *sieving algorithms* with Grover's search by taking into consideration fixed-point arithmetic operations, non-asymptotic Grover's search, the cost of QRAM, and quantum error correction.

1.1 Previous works

Sieving algorithms, introduced by Ajtai, Kumar, and Sivakumar [12, 11], attempt to solve SVP by sampling several vectors and combining them together in order to generate shorter vectors. The sampled vectors are thus repeatedly "sieved" using a norm-reducing operation until a vector with shortest norm remains. The first practical and heuristic sieving algorithm was designed by Nguyen and Vidick [160]. The Nguyen-Vidick sieve (`NVSieve`) solves SVP in a D-dimensional lattice in time $2^{0.415D+o(D)}$ under heuristic assumptions. Shortly after, Micciancio and Voulgaris [156] presented `GaussSieve`, a heuristic sieving algorithm with a time complexity conjectured to be equal to that of `NVSieve`, i.e., $2^{0.415D+o(D)}$, but with better performance in practice. Since then, several new sieving algorithms have been proposed [197, 204, 127, 129, 33, 34, 32]. In particular, heuristic sieves like `NVSieve` and `GaussSieve` have been improved with nearest-neighbour-search methods [104] like locality sensitive hashing (LSH) [57, 20, 21] and locality sensitive filtering (LSF) [33, 32]. These techniques allow to reduce the number of vector comparisons by storing low-dimensional sketches (hashes) such that nearby vectors have a higher chance of sharing the same hash than far away vectors. The asymptotically best classical sieving algorithms are the `NVSieve`/`GaussSieve` enhanced with spherical LSH [129] and `NVSieve`/`GaussSieve` enhanced with spherical LSF [32], which can heuristically solve SVP in time $2^{0.2971D+o(D)}$ and $2^{0.2925D+o(D)}$, respectively. For more on sieving algorithms, see the review [192].

Quantum algorithms for SVP have recently been explored. Laarhoven, Mosca, and van de Pol [130] studied the impact of Grover's search on the asymptotic complexity of various classical sieving algorithms, including `NVSieve` and `GaussSieve`. They concluded that SVP can be heuristically solved on a quantum computer in time $2^{0.2671D+o(D)}$ by employing Grover's search on `NVSieve`/`GaussSieve` with spherical LSH, a $\approx 9\%$ reduction in exponent compared to the classical complexity of [129].

Later, Laarhoven [128] improved the time complexity to $2^{0.2653D+o(D)}$ by employing Grover's search on `NVSieve`/`GaussSieve` with spherical LSF, again leading to a $\approx 9\%$ reducing in exponent compared to its classical counterpart [32]. Chailloux and Loyer [54], on the other hand, presented a modified algorithm in which Grover's search over a filtered list is replaced with a quantum random walk [147]. This brings down the asymptotic time of the quantum algorithm to $2^{0.2570D+o(D)}$. We note that their algorithm still uses Grover's search in the update operation of the quantum random walk. Other works on quantum heuristic sieving algorithms include [118]. Regarding provably correct algorithms, Aggarwal et al. [4] more recently gave a provable quantum algorithm that solves SVP in time $2^{0.95D+o(D)}$ and requires $2^{0.5D+o(D)}$ classical memory and poly(D) qubits. If given access to a QRAM of size $2^{0.293D+o(D)}$, their algorithm requires time $2^{0.835D+o(D)}$ while using the same amount of classical memory and qubits. This improves upon the previously known fastest classical provable algorithm [5].

Beyond asymptotic complexities, Albrecht et al. [14] analysed the cost of quantum algorithms for nearest neighbor search with focus on sieving algorithms. They presented a quantum circuit for performing a simple version of LSF using a *population count* filter, which lets two vectors through the same filter whenever their hashes (using Charikar's LSH scheme [57]) have small Hamming distance. The authors then employed Grover's algorithm inside a quantum amplitude amplification routine [46] to search over the filtered list of nearby vectors to a given vector. By assuming 32 bits of precision, taking quantum arithmetic operations into consideration, disregarding the cost of QRAM, and using a simplified quantum error-correction analysis, Albrecht et al. [14] compared the number of classical and quantum operations employed by three different sieving algorithms: the `NVSieve` [160], the bgj1 specialisation [13] of the Becker-Gama-Joux sieve [33] (which is akin to `NVSieve` with angular LSH [127]), and the `NVSieve` with spherical LSF [32]. They concluded that the number of quantum operations is indeed asymptotically smaller than the number of classical operations, but are comparable at cryptographic dimensions of interest. For example, at dimension $D = 400$, which is roughly the dimension in which SVP has to be solved to be able to break the minimally secure post-quantum cryptographic standards currently being standardised [27], Albrecht et al. [14, Figure 2] estimated that `NVSieve` with spherical LSF (called ListDecodingSearch in their paper) requires either $\approx 10^{42}$ quantum operations or $\approx 10^{43}$ classical operations.

Regarding other works on resource estimations of quantum sieving algorithms, Kim et al. [117] estimated the number of logical qubits and logical quantum gates required by Grover's search on `NVSieve` to solve SVP in lattices of small dimensions. As an example, by ignoring QRAM and quantum error correction, the authors estimated that a *single* Grover's search would require $\approx 7 \cdot 10^7$ logical quantum gates and $\approx 1.5 \cdot 10^6$ logical qubits in dimension $D = 70$ (cf. [117, Table 3]). On the other hand, Prokop et al. [171] proposed a quantum circuit for and studied the resource requirements of a Grover oracle for SVP and analysed how to combine Grover's search with the BKZ algorithm. Beyond sieving algorithms, we briefly mention a variational quantum algorithm proposal with resource estimations for the NISQ era [15] and estimations for quantum enumeration algorithms [28, 39] and for Grover's search attacks on EAS [94, 17, 42, 105] and on SHA-2/SHA-3 [18].

1.2 Our contributions

In this paper, we study how practical quantum speedups for lattice sieves are by performing a precise estimate on the amount of resources required to implement Grover's search on several sieving algorithms. The sieving algorithms considered in this work are the plain `NVSieve` [160] and `GaussSieve` [156] and their enhanced versions with angular/hyperplane LSH (also known as `HashSieve`) [127], with spherical/hypercone LSH (also known as `SphereSieve`) [129], and with spherical/hypercone LSF (also known as `BDGL` sieve) [32], to a total of 8 different sieves. Each of these sieving algorithms perform several Grover's searches per sieving step in order to find lattice vectors that can be combined to yield a new lattice vector with a smaller norm. We compute the amount of physical qubits and time required to perform *all* Grover's searches in a typical instance of the aforementioned sieves. For such, we take into consideration:

1. **Fixed-point quantum arithmetic.** Every entry of a D-dimensional vector is stored using two's-complement representation with $\kappa = 32$ (qu)bits and arithmetic operations on a quantum computer are performed modulo 2^κ. Possible overflows are ignored. We decompose the Grover oracle behind each sieving algorithm into basic arithmetic operations like addition, comparison, and multiplication, and employ quantum circuits for each such arithmetic operation. For quantum addition and comparison, we utilise Gidney's out-of-place quantum adder [85], which has the lowest Toffoli-count of all quantum adders that we are aware of. For quantum multiplication, we utilise a simple decomposition into quantum adders based on schoolbook multiplication that has lower Toffoli-count compared to previous works. A similar construction has appeared in [25] and, very recently, in [141].

2. **Non-asymptotic Grover's search.** It is well known that Grover's search requires $\lfloor \frac{\pi}{4}\sqrt{N/M} \rfloor$ iterations to find one out of M marked elements in a database of size N with high probability if M and N are known beforehand. We do *not* assume to know the number of solutions to any Grover's search within a sieving algorithm. This requires an exponential search Grover's algorithm [43] whose complexity beyond an asymptotic scaling was analysed by Cade, Folkertsma, Niesen, and Weggeman [50] and which we borrow.

3. **Quantum random access memory.** We take into consideration the cost of employing quantum random access memory (QRAM) to quantumly access a classical database within Grover's search. We work exclusively with QRAMs that access *classical data* and consider the circuit implementation from Arunachalam et al. [23] (see also [66]) of the bucket-brigade QRAM architecture [89, 90], which is conceptually simple, has shallow depth, and is noise resilient [99]. We assume that the memory content can be classically rewritten without affecting the QRAM circuit.

4. **Physical architectures.** It is necessary to specify a physical architecture for a general-purpose fault-tolerant quantum computer. Here we assume two different types of architectures: *baseline* architectures with nearest-neighbor logical two-qubit interactions on a 2D grid [138, 78, 55, 56, 41], of which Google's Sycamore processor [24] is an example, and the *active-volume* architecture recently proposed by Litinski and Nickerson [142] that employs a logarithmic number of non-local connections between logical qubits.

5. **Quantum error correction.** Physical quantum computers are heavily affected by noise and a realistic resource estimate should take this into consideration. In this paper we assume an incoherent circuit-level noise model for the physical qubits with error $p_{\text{phy}} = 10^{-5}$. In order to protect against errors, we use surface codes introduced by Kitaev [120, 121] to encode logical qubits, or more precisely, a patch-based surface-code encoding [102]. The time required to measure all surface-code check operators as part of error detecting and correction defines a *code cycle*, which we assume to be 100 ns. The most expensive operations on surface codes are non-Clifford gates like T and Toffoli gates, which can be performed by consuming "magic states" [48] akin to teleportation protocols. We take into consideration space and time overheads to consume magic states by following the framework of [138, 142]. In order to prepare low-error magic states, short error-detecting quantum procedures known as magic state distillation protocols [48, 176] are used. Here we employ the distillation protocols from Litinski [139] which are one of the best we know of. More specifically, we employ a three-level concatenation distillation protocol by using two 15-to-1 punctured Reed-Muller codes [48, 97] followed by a third and final 8-to-CCZ distillation protocol [87] to obtain $|CCZ\rangle$ magic states with errors smaller than 10^{-40}, which are used to perform fault tolerant Toffoli gates. Finally, the time required to perform a layer of measurements, feed the measurement outcomes into a classical decoder, perform a classical decoding algorithm like minimum-weight perfect matching [74, 65] or union-find [64, 63], and use the result to send new instructions to the quantum computer is called *reaction time*. We assume a reaction time of 1 μs. We note that, although the values used here for circuit-level noise, code cycle, and reaction time are not strictly impossible, they are quite optimistic.

6. **Classical hashing operations.** Hashing techniques can be used to decrease the time searching for reducing vectors and require purely classical operations. We take into consideration the amount of time required to classically hash vectors on top of the time required to quantumly search for reducing vectors with Grover algorithm. We break the hashing operations into additions and multiplications and assume that one addition takes 1 cycle/instruction while one multiplication takes 4 cycles/instructions. We consider a 6-GHz-clock-rate single-core classical computer, i.e., it performs $6 \cdot 10^9$ instructions per second. We disregard memory allocation times.

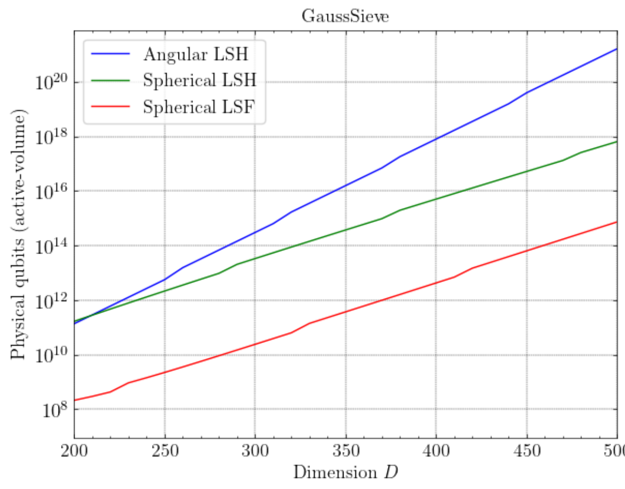
For the sake of comparison, we also consider classical versions of `NVSieve` and `GaussSieve` in which the searching part is perform classically in a sequential manner instead of using Grover algorithm. For such, we decompose the searching operation into basic arithmetic operations like addition and multiplications (this decomposition is the same for the Grover oracle). Similarly to the classical hashing operations, we assume that one addition takes 1 instruction and one multiplication takes 4 instructions. We consider a 6-GHz-clock-rate single-core classical computer.

Although resource estimates as comprehensive as ours have been carried out under similar considerations for algorithms like Shor's [86, 140], we are not aware of similar results on sieving (or enumeration) algorithms. The work of Albrecht et al. [14] is the closest to our results, but they fall short of considering QRAM and conducting a more rigorous analysis on quantum error correction. As an example, the scripts provided by Gidney and Ekerå [86] and adapted by Albrecht et al. consider two-level distillation protocols which, although enough in the context of Shor's algorithm, cannot produce magic states with small enough errors for sieving algorithms in high dimensions. A three or four-level distillation protocol is required to reach errors below 10^{-40} or even 10^{-50}.

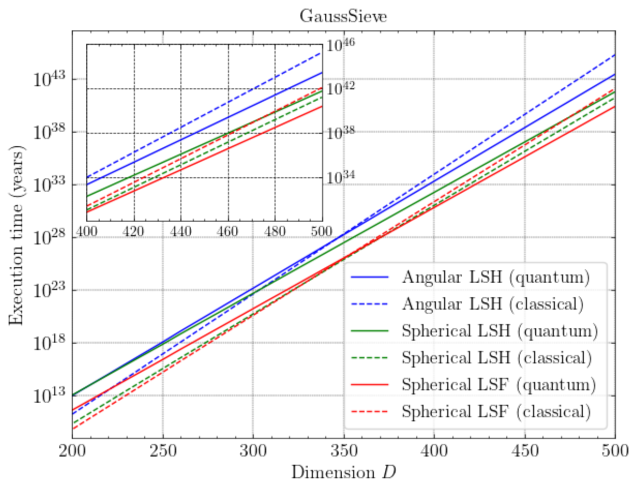
Since `NVSieve` and `GaussSieve` are inherently randomised algorithms, we carried out the resource estimates under heuristic assumptions on the value of internal parameters of these sieves. As examples, we assume that the initial list size in `NVSieve` is $D \cdot 2^{0.2352D+0.102\log_2 D+2.45}$ as numerically computed by Nguyen and Vidick [160], while the maximum list size in `GaussSieve` is $2^{0.193D+2.325}$ as calculated by us and similarly reported by Mariano et al. [149]. The number of sieving steps in `GaussSieve` has been reported to grow as $2^{0.283D+0.335}$ by Mariano et al. [149] and independently checked by us. We refer the reader to Section 8.2 for a complete list of assumptions on the average performance of `NVSieve` and `GaussSieve`. On the other hand, the use of hashing techniques (LSH and LSF) introduces two tunable parameters: the size of the hash space and the number of hash tables. The values used for these parameters are highly heuristic in practice, while in asymptotic analyses they are chosen so to guarantee that nearby vectors collide (have the same hash) in at least one hash table with high probability and to balance out the time spent hashing and the time spent searching for reducing vectors. Here we follow a (slightly more detailed) version of the asymptotic analysis. To be more precise, we set the parameters in order to balance the classical hashing time and the quantum searching time by *ignoring overall complexity constant factors*, meaning that classically hashing a list of certain size would be roughly as costly as quantumly searching the same list. Although not an entirely realistic assumption, it is optimistic in that it lessens the computational burden on hashing. We leave a more detail analysis on the hashing parameters for a future work.

Our main results are condensed in Figure 1, where we show the amount of physical qubits and time required by `GaussSieve` with LSH/LSF as a function of the lattice dimension D. We consider an active-volume architecture and omit results for the `NVSieve` for now as `GaussSieve` has a better performance. The number of physical qubits from Figure 1a is the number of physical qubits required to run the largest Grover's search in `GaussSieve`, since physical qubits can be reused in different searches. On the other hand, Figure 1 shows the time required to execute both a classical and a quantum version of `GaussSieve`, i.e., where the searching is performed either classically or via Grover's search. More precisely, the execution time of the classical `GaussSieve` is the sum of all searching and hashing operations, while the execution time of the quantum `GaussSieve` is the time required to *sequentially* execute all Grover's searches plus the time required to classically hash all vectors.

At dimensions of cryptographic interest, e.g., at dimension $D = 400$, `GaussSieve` with spherical LSF requires $\approx 10^{13}$ physical qubits to solve SVP in $\approx 10^{31}$ years. As shown in Section 8.3, most of the



(a) Active-volume physical qubits of `GaussSieve`



(b) Execution time of `GaussSieve`

Figure 1: Number of physical qubits and execution time of all Grover's searches in `GaussSieve` with LSH/LSF as a function of the lattice dimension D. We assume an underlying active-volume physical architecture. The execution time is the sum of the time spent searching for pairs of reducing vectors (either quantumly or classically) and the classical time spent hashing.

physical qubits are coming from the use of a bucket-brigade-style QRAM, since it requires a number of logical qubits roughly equal to the size of the accessed database. The total time comes mostly from the quantum arithmetic circuits and the fact that Grover's search requires, at the end of the day, a deep circuit. A classical version of `GaussSieve` with spherical LSF also requires roughly the same amount of time to solve SVP.

Figure 1 paints a pessimistic scenario for quantum sieving algorithms, with the number of physical qubits surpassing modern transistor counts by a few orders of magnitude and a total execution time comparable to their classical counterpart and greater than the age of the universe. While Albrecht et al. [14] compared the number of (arithmetic) classical and quantum operations, which is not ideal as the cost of various elementary operations can vary significantly, here we resolve both classical and quantum operations into actual execution times. The end result as seen in Figure 1b is a small quantum advantage for dimensions beyond 400: at $D = 500$, Grover's search provides a speedup by roughly two orders of magnitude.

We stress that the above numbers ignore all the memory fetch operations, which although should worsen both classical and quantum runtimes, will most likely impact the quantum one more severely since, as we argue in Section 7, the use of hashing techniques yields lists of candidate vectors that require several RAM calls to be accessed via QRAM and thus be searched with Grover algorithm. Moreover, classical searching operations can be more easily parallelised than Grover's search [203], in the sense that F parallel Grover algorithms running on F separate search spaces have a total width that is larger by a factor of F compared to a single Grover algorithm on the whole search space while only reducing the depth by a factor of $\sqrt{F}$.

It is expected that several assumptions, numbers, and protocols used in this work will become dated in a few years and several new results on circuit design, quantum error correction, and QRAM will be discovered (and a few new improvements have indeed been posted online by the time this manuscript was been finalised [158, 88, 198]), but we believe that the overall message remains that Grover's search (and quadratic improvements for that matter) offers very little advantage over classical search in sieving algorithms at dimensions of cryptographic interest. Any considerable speedups will occur on dimensions far larger than the ones needed for most cryptographic purposes or require significant breakthroughs in theoretical protocols and hardware development.

The remainder of the paper is organised as follows. In Section 2 we review basic concepts from quantum computation and hashing techniques like LSH and LSF. In Section 3 we review several key ideas from quantum error correction like surface codes, baseline and active-volume architectures, and magic state distillation protocols. Section 4 covers all quantum arithmetic circuits employed in our paper. Section 5 reviews Grover's search algorithm, while Section 6 reviews the bucket-brigade QRAM. In Section 7 we describe the `NVSieve` and `GaussSieve` with and without LSH/LSF and construct the Grover oracles for them. In Section 8 we perform our resource estimation analysis. This section is divided into a few parts: Section 8.1 describes how the resource estimation is performed for the example when $D = 400$; Section 8.2 describes our main results; Section 8.3 analyses the cost of QRAM; Section 8.4 explores the impact of depth restrictions as proposed by NIST post-quantum cryptography standardisation process [162]. Finally, we conclude in Section 9. The source code and data can be found in [1].

2 Preliminaries

Given $n \in \mathbb{N} := \{1, 2, \dots\}$, define $[n] := \{1, \dots, n\}$. Let $\mathsf{X} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $\mathsf{Y} = \left(\begin{smallmatrix} 0 & -i \\ i & 0 \end{smallmatrix}\right)$, and $\mathsf{Z} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$ be the usual Pauli matrices and I_n the n-dimensional identity matrix. We shall refer to I_n simply as I when the dimension is clear from context. Let $\mathbf{1}[\text{clause}] \in \{0, 1\}$ be the indicator function that equals 1 if the clause is true and 0 otherwise. Given vectors $\mathbf{v}$, $\mathbf{w} \in \mathbb{R}^D$, let $\|\mathbf{v}\| := (\sum_{i=1}^D v_i^2)^{1/2}$ be the Euclidean norm of $\mathbf{v}$, $\theta(\mathbf{v}, \mathbf{w})$ the angle between $\mathbf{v}$ and $\mathbf{w}$, and $\langle \mathbf{v}, \mathbf{w} \rangle := \sum_{i=1}^D v_i w_i$ their inner product. Let $\Gamma(z)$ be the gamma function. We denote by $\mathcal{S}^{D-1} := \{\mathbf{v} \in \mathbb{R}^D : \|\mathbf{v}\| = 1\}$ the D-dimensional unit hypersphere and by $\mathcal{H}_{\mathbf{v},\alpha} := \{\mathbf{x} \in \mathbb{R}^D : \langle \mathbf{v}, \mathbf{x} \rangle \geq \alpha\}$ the half-spaces, where $\mathbf{v} \in \mathcal{S}^{D-1}$. Let $\mathcal{C}_D(\alpha)$ be the measure of the spherical cap $\mathcal{C}_{\mathbf{v},\alpha} := \mathcal{S}^{D-1} \cap \mathcal{H}_{\mathbf{v},\alpha}$ and $\mathcal{W}_D(\alpha, \beta, \theta)$ be the measure of the spherical wedge $\mathcal{W}_{\mathbf{v},\alpha,\mathbf{w},\beta} := \mathcal{S}^{D-1} \cap \mathcal{H}_{\mathbf{v},\alpha} \cap \mathcal{H}_{\mathbf{w},\beta}$, where $\mathbf{v}, \mathbf{w} \in \mathcal{S}^{D-1}$ with $\langle \mathbf{v}, \mathbf{w} \rangle = \cos\theta$. We shall need the next known facts.

Fact 1 ([128, Lemma 10.7])**.** *The probability density function $\Theta_{[\theta_1,\theta_2]}(\theta)$ of angles between vectors $\mathbf{v}, \mathbf{w} \in \mathcal{S}^{D-1}$ drawn at random from the unit sphere and such that $\theta_1 \leq \theta(\mathbf{v}, \mathbf{w}) \leq \theta_2$ is*

$$\Theta_{[\theta_1,\theta_2]}(\theta) = \frac{\sin^{D-2}\theta}{\int_{\theta_1}^{\theta_2} \sin^{D-2}\phi \, \mathrm{d}\phi}.$$

Fact 2 ([136])**.** *Let $\mathbf{v} \in \mathcal{S}^{D-1}$ and $\alpha \in (0, 1)$. The measure $\mathcal{C}_D(\alpha)$ of the spherical cap $\mathcal{C}_{\mathbf{v},\alpha}$ is*

$$\mathcal{C}_D(\alpha) := \frac{\mu(\mathcal{C}_{\mathbf{v},\alpha})}{\mu(\mathcal{S}^{D-1})} = \frac{1}{\sqrt{\pi}} \frac{\Gamma(\frac{D}{2})}{\Gamma(\frac{D-1}{2})} \int_0^{\arccos\alpha} \sin^{D-2}\phi \, \mathrm{d}\phi.$$

Fact 3 ([132, Case 8])**.** *Let $\mathbf{v}, \mathbf{w} \in \mathcal{S}^{D-1}$ with $\langle \mathbf{v}, \mathbf{w} \rangle = \cos\theta$. Let $\alpha, \beta \in (0, 1)$ such that $\theta < \arccos\alpha + \arccos\beta$ and $(\alpha - \beta\cos\theta)(\beta - \alpha\cos\theta) > 0$. Define $\theta^* \in (0, \frac{\pi}{2})$ by $\tan\theta^* = \alpha/(\beta\sin\theta) - 1/\tan\theta$. The measure $\mathcal{W}_D(\alpha, \beta, \theta)$ of the spherical wedge $\mathcal{W}_{\mathbf{v},\alpha,\mathbf{w},\beta}$ is*

$$\mathcal{W}_D(\alpha, \beta, \theta) := \frac{\mu(\mathcal{W}_{\mathbf{v},\alpha,\mathbf{w},\beta})}{\mu(\mathcal{S}^{D-1})} = J_D(\theta^*, \arccos\beta) + J_D(\theta - \theta^*, \arccos\alpha),$$

where

$$J_D(\theta_1, \theta_2) := \frac{1}{\sqrt{\pi}} \frac{\Gamma(\frac{D}{2})}{\Gamma(\frac{D-1}{2})} \int_{\theta_1}^{\theta_2} \mathcal{C}_{D-1}\left(\arccos\left(\frac{\tan\theta_1}{\tan\phi}\right)\right) \sin^{D-2}\phi \, \mathrm{d}\phi.$$

2.1 Quantum computing

We assume the reader is somewhat familiar with quantum computing. The quantum state of a quantum system is described by a vector from a Hilbert space $\mathscr{H}$, i.e., a complex vector space with inner product structure. A qubit, the quantum equivalent of a bit, is a quantum system described by a vector in $\mathscr{H} \cong \mathbb{C}^2$, while an n-qubit system is described by a vector $|\psi\rangle$ in $\mathscr{H} \cong \mathbb{C}^{2^n}$. Equivalently, an n-qubit quantum system can also be described by a density matrix $\rho \in \mathbb{C}^{2^n\times 2^n}$, i.e., a semi-definite positive matrix with unit trace. The evolution of a quantum state $|\psi\rangle \in \mathbb{C}^{2^n}$ is described by a unitary operator $\mathsf{U} \in \mathbb{C}^{2^n\times 2^n}$, $\mathsf{U}\mathsf{U}^\dagger = \mathsf{I}$ where $\mathsf{U}^\dagger$ is the Hermitian conjugate of U. A unitary operator is also referred to as a quantum gate. In order to extract classical information from a quantum system, a quantum measurement is usually performed. A quantum measurement is expressed as a positive operator-valued measure (POVM), i.e., a set $\{\mathsf{E}_m\}_m$ of positive operators $\mathsf{E}_m \succ 0$ that sum to identity, $\sum_m \mathsf{E}_m = \mathsf{I}$. The probability of measuring E_m on $|\psi\rangle$ is $p_m = \langle\psi|\mathsf{E}_m|\psi\rangle$. A quantum circuit is a sequence of quantum gates acting on a set of qubits. At the end of the circuit, a measurement is performed and a classical outcome is observed. We refer the reader to [161, 199] for more information.

There are a few sets of universal gates that can serve as building blocks for any quantum circuit. One of the most common is the Clifford+T gate set comprising the one and two-qubit gates

$$\mathsf{H} = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix},\ \mathsf{S} = \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix},\ \mathsf{T} = \begin{pmatrix}1 & 0\\ 0 & e^{i\pi/4}\end{pmatrix},\ \mathsf{CNOT} = \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}.$$

Here, $\mathsf{H}, \mathsf{CNOT}, \mathsf{S}$ are Clifford gates, while the T gate is a non-Clifford gate (it does not normalise the Pauli group). The Clifford+T gate set $\{\mathsf{H}, \mathsf{S}, \mathsf{T}, \mathsf{CNOT}\}$ is universal [67, 44], meaning that any quantum circuit can be written in terms of its elements as accurately as required. Another universal gate set is $\{\mathsf{H}, \mathsf{S}, \mathsf{CNOT}, \mathsf{Toffoli}\}$ [187], where

$$\mathsf{Toffoli} = \begin{pmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&1&0\end{pmatrix}.$$

Here, $\mathsf{Toffoli}$ is a non-Clifford gate. Define also the CZ and CCZ gates as $\mathsf{CZ} = (\mathsf{I}_2 \otimes \mathsf{H})\mathsf{CNOT}(\mathsf{I}_2 \otimes \mathsf{H})$ and $\mathsf{CCZ} = (\mathsf{I}_4 \otimes \mathsf{H})\mathsf{Toffoli}(\mathsf{I}_4 \otimes \mathsf{H})$, respectively. In this work, we shall focus on the $\{\mathsf{H}, \mathsf{S}, \mathsf{CNOT}, \mathsf{Toffoli}\}$ universal gate set, as all of our circuits can be naturally decomposed using these gates. We shall also consider the CCZ gate to have the same cost as the $\mathsf{Toffoli}$ gate and will count them as a single resource.

By *ancillary qubits* (or simply *ancillae*) we mean qubits that can be re-used across computation, so that a gate U_2 can use ancillae from some previous gate U_1. This means that if two gates U_1 and U_2 use c_1 and c_2 ancillae, respectively, then the joint gate $\mathsf{U}_1\mathsf{U}_2$ requires $\max(c_1, c_2)$ ancillae. By *dirty ancillae* we mean auxiliary qubits employed in a quantum gate that are left entangled with other qubits and therefore cannot be reused in later computations afresh. This means that if two gates U_1 and U_2 use c_1 and c_2 dirty ancillae, respectively, then the joint gate $\mathsf{U}_1\mathsf{U}_2$ requires $c_1 + c_2$ dirty ancillae. We will routinely keep dirty ancillae after some computation to facilitate its uncomputation at a later time.

By $\mathsf{C}^{(k)}$-X we mean an X gate controlled on k qubits, i.e., an X gate is applied conditioned on all k qubits being on the $|1\rangle$ state. This means that $\mathsf{C}^{(1)}$-$\mathsf{X} = \mathsf{CNOT}$ and $\mathsf{C}^{(2)}$-$\mathsf{X} = \mathsf{Toffoli}$. It is possible to decompose $\mathsf{C}^{(k)}$-X into $\mathsf{Toffoli}$ gates as summarised in the next well-known result.

Fact 4 (Multi-controlled Toffoli)**.** *The multi-controlled* Toffoli *gate* $\mathsf{C}^{(k)}$-X *with* $k > 1$ *controls can be implemented using* $k-1$ Toffoli *gates and* $k-2$ *ancillae.*

2.2 Locality-sensitive hashing and locality-sensitive filtering

In this work, we consider lattice sieving algorithms. These are algorithms that start with (an exponentially) large list of lattice vectors consisting of long vectors and use it to find shorter lattice vectors. If the length of the vectors in the initial list is roughly the same, then this can be done by finding nearby lattice vectors in the list, since their difference would be a shorter lattice vector. More precisely, we would like to

$$\text{find vectors } \mathbf{v}, \mathbf{w} \text{ from a list } L \text{ such that } \|\mathbf{v} \pm \mathbf{w}\| \leq \max\{\|\mathbf{v}\|, \|\mathbf{w}\|\},$$

which is equivalent to

$$\text{find vectors } \mathbf{v}, \mathbf{w} \text{ from a list } L \text{ such that } \theta(\mathbf{v}, \pm\mathbf{w}) \leq \pi/3$$

if $\|\mathbf{v}\| \approx \|\mathbf{w}\|$. The above problem can naturally be framed as a *nearest neighbour search*. In the nearest neighbour search, a list of D-dimensional vectors $L = \{\mathbf{w}_1, \dots, \mathbf{w}_N\} \subset \mathbb{R}^D$ is given and the task is to preprocess L in such a way that given a new vector $\mathbf{v} \notin L$, it is possible to efficiently find an element $\mathbf{w} \in L$ close(st) to $\mathbf{v}$. *Locality-sensitive hashing* (LSH) is a well-known technique to speed up nearest neighbour search and it makes use of locality-sensitive hash functions [104]. A locality-sensitive hash function $h(\cdot)$ projects a D-dimensional vector into a low-dimension sketch and has the property that nearby vectors have a higher probability of collision than far away vectors. This sketch can then be used to bucket vectors in L such that the vectors in the same bucket are close and hence speed up the search. A family of hash functions $\mathcal{H} = \{h : \mathbb{R}^D \to U \subset \mathbb{N}\}$ is characterised by the collision probability

$$p(\theta) := \Pr_{h\sim\mathcal{H}}[h(\mathbf{v}) = h(\mathbf{w}) \mid \mathbf{v}, \mathbf{w} \in \mathcal{S}^{D-1}, \langle \mathbf{v}, \mathbf{w}\rangle = \cos\theta],$$

where $h \sim \mathcal{H}$ means a hash function h uniformly picked over $\mathcal{H}$.

Another well-known technique is *locality-sensitive filtering* (LSF) [32], which employs a filter that maps a vector to a binary value: a vector either passes a filter or not. A filter that a vector $\mathbf{v}$ passes through is called a *relevant filter* for $\mathbf{v}$. Applied to a list L, a filter f maps L to an output filtered list $L_f \subset L$ of points that survive the filter. The idea is to choose a filter that yields an output list L_f of only nearby vectors. A family of filter functions $\mathcal{F} = \{f : \mathbb{R}^D \to \{0, 1\}\}$ is characterised by the collision probability

$$p(\theta) := \Pr_{f\sim\mathcal{F}}[\mathbf{v}, \mathbf{v} \in L_f \mid \mathbf{v}, \mathbf{w} \in \mathcal{S}^{D-1}, \langle \mathbf{v}, \mathbf{w}\rangle = \cos\theta],$$

where $f \sim \mathcal{F}$ means a filter function f uniformly picked over $\mathcal{F}$. We note that while $p(0) = 1$ for hash families, the same is not true for most filter families, since in general the collision probability of $\mathbf{v}$ with itself is $p(0) < 1$.

A hash/filter family with $p(\theta_1) \gg p(\theta_2)$ can efficiently distinguish nearby vectors at angle θ_1 from distant vectors at angle θ_2 by looking at their hash/filter values. The existence of hash/filter families with $p(\theta_1) \approx 1$ and $p(\theta_2) \approx 0$ is, however, not straightforward. A common technique is to first construct a hash/filter family with $p(\theta_1) \approx p(\theta_2)$ and use a series of AND- and OR-compositions to amplify the gap between $p(\theta_1)$ and $p(\theta_2)$ and obtain a new hash/filter family with $p'(\theta_1) > p(\theta_1)$ and $p'(\theta_2) < p(\theta_2)$.

AND-composition. Given a hash family $\mathcal{H}$ with collision probability $p(\theta)$, it is possible to construct a hash family $\mathcal{H}' = \mathcal{H}^k$ with collision probability $p(\theta)^k$ by taking k different and pairwise independent hash functions $h_1, \dots, h_k \in \mathcal{H}$ and defining $h \in \mathcal{H}'$ such that $h(\mathbf{v}) = (h_1(\mathbf{v}), \dots, h_k(\mathbf{v}))$. Clearly $h(\mathbf{v}) = h(\mathbf{w})$ if and only if $h_i(\mathbf{v}) = h_i(\mathbf{w})$ for all $i \in [k]$, and thus $p'(\theta) = p(\theta)^k$. Similarly for a filter family $\mathcal{F}$.

OR-composition. Given a hash family $\mathcal{H}$ with collision probability $p(\theta)$, it is possible to construct a hash family $\mathcal{H}'$ with collision probability $1 - (1 - p(\theta))^t$ by taking t different and pairwise independent hash functions $h_1, \dots, h_t \in \mathcal{H}$ and defining $h \in \mathcal{H}'$ by the relation $h(\mathbf{v}) = h(\mathbf{w})$ if and only if $h_i(\mathbf{v}) = h_i(\mathbf{w})$ for some $i \in [t]$. Clearly $h(\mathbf{v}) \neq h(\mathbf{w})$ if and only if $h_i(\mathbf{v}) \neq h_i(\mathbf{w})$ for all $i \in [t]$, and thus $1 - p'(\theta) = (1 - p(\theta))^t$. Similarly for a filter family $\mathcal{F}$.

Suitable hash/filter families together with AND and OR-compositions can be used to find nearest neighbors as first described by Indyk and Motwani [104]. The idea is to choose $t \cdot k$ hash functions $h_{i,j} \in \mathcal{H}$ from some hash family $\mathcal{H}$ and use the AND-composition to combine k of them at a time to build t new hash functions $h_1, \ldots, h_t$, where $h_i(\cdot) = (h_{i,1}(\cdot), \ldots, h_{i,k}(\cdot))$ for $i \in [t]$. Then, given the list L, we build t different hash tables $\mathcal{T}_1, \ldots, \mathcal{T}_t$ and for each hash table $\mathcal{T}_i$ we insert a vector $\mathbf{w} \in L$ from the list into the bucket labelled by $h_i(\mathbf{w})$. This means that all the vectors from L are inserted into an appropriate bucket in each hash table. Finally, given a target vector $\mathbf{v}$, we compute its t hash images $h_1(\mathbf{v}), \ldots, h_t(\mathbf{v})$ and look only for candidate vectors in the bucket labelled $h_i(\mathbf{v})$ in hash table $\mathcal{T}_i$, for all $i \in [t]$ (OR-composition). In other words, we consider only the vectors that collide with $\mathbf{v}$ in at least one of the hash tables. A similar idea applies to filter families. A vector $\mathbf{v}$ is inserted into a filtered bucket $\mathcal{B}_i$ if and only if it survives the concatenated filter f_i made out of filters $f_{i,1}, \ldots, f_{i,k}$, for $i \in [t]$.

2.2.1 Angular LSH

A famous hash family is the angular (or hyperplane) locality-sensitive hash method of Charikar [57], which, as we will see in Section 7, can be used to improve sieving algorithms [127]. Charikar proposed the following hash family $\mathcal{H}_{\text{ang}}$,

$$\mathcal{H}_{\text{ang}} = \{h_{\mathbf{a}} : \mathbb{R}^D \to \{0,1\} \mid \mathbf{a} \in \mathcal{S}^{D-1}\}, \qquad h_{\mathbf{a}}(\mathbf{v}) = \begin{cases} 1 & \text{if } \langle \mathbf{a}, \mathbf{v} \rangle \geq 0, \\ 0 & \text{if } \langle \mathbf{a}, \mathbf{v} \rangle < 0. \end{cases}$$

The vector $\mathbf{a}$ defining the hash function $h_{\mathbf{a}}$ also defines a hyperplane (for which $\mathbf{a}$ is a normal vector), and $h_{\mathbf{a}}$ maps the two regions separated by the hyperplane onto different bits. Charikar proved [57] that the probability of collision is $p(\theta) = 1 - \theta/\pi$, which can be seen from the fact that two vectors $\mathbf{v}, \mathbf{w}$ define a two-dimensional plane and these two vectors are mapped onto different hashes if a random line (the intersection between this plane and the hyperplane defined by $\mathbf{a}$) separates $\mathbf{v}$ and $\mathbf{w}$.

Under the angular hash family $\mathcal{H}_{\text{ang}}$, consider t hash tables, each with 2^k hash buckets, constructed via AND and OR-compositions with randomly sampled hash functions $h_{i,j} \in \mathcal{H}_{\text{ang}}$ as previously described. It is possible to calculate the average probability p_1^* that two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^D$ with $\theta(\mathbf{v}, \mathbf{w}) \leq \pi/3$ collide in at least one of the t hash tables:

$$p_1^* = \Pr_{h_{i,j} \sim \mathcal{H}_{\text{ang}}}[\exists i \in [t], h_i(\mathbf{v}) = h_i(\mathbf{w}) \mid \theta(\mathbf{v}, \mathbf{w}) \leq \pi/3] = \int_0^{\frac{\pi}{3}} \Theta_{[0,\frac{\pi}{3}]}(\theta)\big(1 - (1 - (1 - \theta/\pi)^k)^t\big)\mathrm{d}\theta.$$

It can be shown (see [128, Lemma 10.5]) that $p_1^* \geq 1 - \varepsilon$ if $k = \log_{3/2} t - \log_{3/2} \ln(1/\varepsilon)$. On the other hand, the average probability p_2^* that two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^D$ with $\theta(\mathbf{v}, \mathbf{w}) > \pi/3$ collide in at least one of the t hash tables is

$$p_2^* = \Pr_{h_{i,j} \sim \mathcal{H}_{\text{ang}}}[\exists i \in [t], h_i(\mathbf{v}) = h_i(\mathbf{w}) \mid \theta(\mathbf{v}, \mathbf{w}) > \pi/3] = \int_{\frac{\pi}{3}}^{\frac{\pi}{2}} \Theta_{[\frac{\pi}{3},\frac{\pi}{2}]}(\theta)\big(1 - (1 - (1 - \theta/\pi)^k)^t\big)\mathrm{d}\theta. \quad (1)$$

It can be shown (see [128, Lemma 10.8]) that $p_2^* \leq t \cdot 2^{-\beta D + o(D)}$ if $k = \log_{3/2} t + O(1)$, where

$$\beta = -\max_{\theta \in (\frac{\pi}{3},\frac{\pi}{2})} \left\{ \log_2 \sin\theta + \frac{\log_2 t}{D \log_2(3/2)} \log_2(1 - \theta/\pi) \right\} > 0. \quad (2)$$

Ultimately, the choice for t will depend on the balance between the time hashing and the time searching, as we shall see in Section 7.

2.2.2 Spherical LSH

Another important hash family that can be used to improve sieving algorithms [129] is the spherical LSH proposed by Andoni et al. [20, 21]. The spherical LSH partitions the unit sphere $\mathcal{S}^{D-1}$ by first sampling $u = 2^{\Theta(\sqrt{D})}$ vectors $\mathbf{g}_1, \dots, \mathbf{g}_u \in \mathbb{R}^D$ from a standard D-dimensional Gaussian distribution $\mathcal{N}(0,1)^D$. A hash region $\mathcal{R}_i$ is then associated to each $\mathbf{g}_i$ as

$$\mathcal{R}_i = \{\mathbf{x} \in \mathcal{S}^{D-1} : \langle \mathbf{x}, \mathbf{g}_i \rangle \geq D^{1/4}\} \setminus \bigcup_{j=1}^{i-1} \mathcal{R}_j, \qquad \forall i \in [u].$$

This procedure sequentially "carves" spherical caps of radius $\sqrt{2} - o(1)$. The hash of a vector $\mathbf{v}$ is given by the index of the region $\mathcal{R}_i$ it lies in. Moreover, the choice of $u = 2^{\Theta(\sqrt{D})}$ guarantees that the unit sphere is entirely covered by the hash regions with high probability since each hash region covers a fraction $2^{-\Theta(\sqrt{D})}$ of the sphere. Indeed, $\Pr_{\mathbf{g}\sim\mathcal{N}(0,1)^D}[\langle \mathbf{x}, \mathbf{g} \rangle \geq D^{1/4}] \geq (2\pi)^{-1/2}(D^{-1/4} - D^{-3/4})e^{-\sqrt{D}/2}$ for any fixed point $\mathbf{x} \in \mathcal{S}^{D-1}$ [113], and by following the argument in [22, Appendix A.3], $u = 2^{\sqrt{D}}$ hash regions is enough to cover the unit sphere with failure probability super-exponentially small in D. Andoni et al. [20, 21] proved that the collision probability for the spherical hash family $\mathcal{H}_{\text{sph}}$ is

$$p(\theta) = \exp\left(-\frac{\sqrt{D}}{2}\tan^2\left(\frac{\theta}{2}\right)(1 + o(1))\right).$$

Under the spherical hash family $\mathcal{H}_{\text{sph}}$ with randomly sampled hash functions $h_{i,j} \in \mathcal{H}_{\text{sph}}$, the average probability p_1^* that two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^D$ with $\theta(\mathbf{v}, \mathbf{w}) \leq \pi/3$ collide in at least one of t hash tables is

$$p_1^* = \Pr_{h_{i,j}\sim\mathcal{H}_{\text{sph}}}[\mathbf{v}, \mathbf{w} \text{ collide} \mid \theta(\mathbf{v}, \mathbf{w}) \leq \pi/3] = \int_0^{\frac{\pi}{3}} \Theta_{[0,\frac{\pi}{3}]}(\theta)\left(1 - \left(1 - e^{-\frac{k\sqrt{D}}{2}\tan^2\left(\frac{\theta}{2}\right)(1+o(1))}\right)^t\right)\mathrm{d}\theta.$$

It can be shown (see [128, Lemma 11.5]) that $p_1^* \geq 1 - \varepsilon$ if $k = 6(\ln t - \ln\ln(1/\varepsilon))/\sqrt{D}$. On the other hand, the average probability p_2^* that two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^D$ with $\theta(\mathbf{v}, \mathbf{w}) > \pi/3$ collide in at least one of t hash tables is

$$p_2^* = \Pr_{h_{i,j}\sim\mathcal{H}_{\text{sph}}}[\mathbf{v}, \mathbf{w} \text{ collide} \mid \theta(\mathbf{v}, \mathbf{w}) > \pi/3] = \int_{\frac{\pi}{3}}^{\frac{\pi}{2}} \Theta_{[\frac{\pi}{3},\frac{\pi}{2}]}(\theta)\left(1 - \left(1 - e^{-\frac{k\sqrt{D}}{2}\tan^2\left(\frac{\theta}{2}\right)(1+o(1))}\right)^t\right)\mathrm{d}\theta. \quad (3)$$

It can be shown (see [128, Lemma 11.6]) that $p_2^* \leq 2^{-\beta D + o(D)}$ if $k = 6\ln(t)/\sqrt{D} + o(1)$, where

$$\beta = -\max_{\theta\in(\frac{\pi}{3},\frac{\pi}{2})}\left\{\log_2 \sin\theta - \left(3\tan^2\left(\frac{\theta}{2}\right) - 1\right)\frac{\log_2 t}{D}\right\} > 0. \quad (4)$$

2.2.3 Spherical LSF

Becker et al. [32] proposed the spherical LSF family akin to spherical LSH. In spherical LSF, a filter is constructed by drawing a random $\mathbf{a} \in \mathcal{S}^{D-1}$ and a vector $\mathbf{v}$ passes the filter if $\langle \mathbf{a}, \mathbf{v} \rangle \geq \alpha$ for some parameter $\alpha > 0$. In other words,

$$\mathcal{F}_{\text{sph}} = \{f_{\mathbf{a}} : \mathbb{R}^D \to \{0,1\} \mid \mathbf{a} \in \mathcal{S}^{D-1}\}, \qquad f_{\mathbf{a}}(\mathbf{v}) = \begin{cases} 1 & \text{if } \langle \mathbf{a}, \mathbf{v} \rangle \geq \alpha, \\ 0 & \text{if } \langle \mathbf{a}, \mathbf{v} \rangle < \alpha. \end{cases}$$

As shown by Becker et al. [32], the collision probability for the spherical filter family $\mathcal{F}_{\text{sph}}$ is

$$p(\theta) = \mathcal{W}_D(\alpha, \alpha, \theta) = \exp\left(\frac{D}{2}\ln\left(1 - \frac{2\alpha^2}{1 + \cos\theta}\right)(1 + o(1))\right), \quad (5)$$

while the collision probability of a vector with itself is

$$p(0) = \mathcal{C}_D(\alpha) = \exp\left(\frac{D}{2}\ln(1 - \alpha^2)(1 + o(1))\right).$$

Under the spherical filter family $\mathcal{F}_{\text{sph}}$ with randomly sampled filters $f_{i,j} \in \mathcal{F}_{\text{sph}}$, the average probability p_1^* that two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^D$ with $\theta(\mathbf{v}, \mathbf{w}) \leq \pi/3$ collide in at least one of t filters is

$$p_1^* = \Pr_{f_{i,j} \sim \mathcal{F}_{\text{sph}}}[\exists i \in [t], \mathbf{v}, \mathbf{w} \in L_{f_i} \mid \theta(\mathbf{v}, \mathbf{w}) \leq \pi/3] = \int_0^{\frac{\pi}{3}} \Theta_{[0,\frac{\pi}{3}]}(\theta)(1 - (1 - \mathcal{W}_D(\alpha, \alpha, \theta)^k)^t)\mathrm{d}\theta. \quad (6)$$

Since $\mathcal{W}_D(\alpha, \alpha, \theta)$ is decreasing in θ, it is not hard to see that $p_1^* \geq 1-(1-\mathcal{W}_D(\alpha, \alpha, \pi/3)^k)^t$. Therefore, $p_1^* \geq 1 - \varepsilon$ if $t \geq \ln(1/\varepsilon)/\ln\big(1/(1 - \mathcal{W}_D(\alpha, \alpha, \pi/3)^k)\big)$. Regarding the choice for k, the trivial lower bound $k \geq 1$ leads to an upper bound on α, which is normally the optimal choice, see [32] for more information. This means that we shall take $k = 1$ in the above expressions.

LSF methods usually yield better asymptotic complexities when it comes to sieving algorithms, as shown in Section 7. However, a crucial assumption for the use of filter families over hash families is the existence of an efficient oracle that identifies any of the concatenated filters a vector passes through in time proportional to the number of relevant filters out of all concatenated filters. Becker et al. [32] developed such an oracle, called `EfficientListDecoding`, by employing random product codes to efficiently obtain the set of relevant filters, which only mildly affects the overall complexities. The complexity of their oracle is summarised below.

Fact 5 ([32, Lemma 5.1]). *Let $t = 2^{\Omega(D)}$ be the number of filter buckets. There is an algorithm that returns the set of filters that a given vectors passes in average time $O(\log_2 D \cdot t \cdot \mathcal{C}_D(\alpha))$ by mainly visiting at most $2\log_2 D \cdot t \cdot \mathcal{C}_D(\alpha)$ nodes for a pruned enumeration.*

3 Quantum error correction

Quantum circuits are usually described on a logical level by applying logical gates onto logical qubits. If one wants to implement a quantum circuit in actual physical devices, then noise should be taken into consideration. This is not only valid for classical devices, but especially true for quantum computers, where exquisite control of quantum systems is severely affected by noise. One of the greatest breakthroughs of the 90s was the realisation that redundancy could also be introduced into quantum systems to protect them against several types of noise, and therefore quantum error-correction codes exist. Starting with Shor's nine-qubit code [184], several simple quantum error-correction codes were soon discovered, e.g., Steane's seven-qubit code [189], the five-qubit code [37, 131], and the CSS (Calderbank-Shor-Steane) codes [51, 190]. All these codes are examples of stabiliser codes, i.e., quantum error-correction codes based on the stabiliser formalism invented by Gottessman [92, 93]. In any quantum error-correction code, a set of *physical* qubits are entangled in particular states and these joint states are interpreted as *logical* qubits. As an example, in Shor's code [184] $|0_L\rangle$ is encoded as $(|000\rangle + |111\rangle)^{\otimes 3}/2\sqrt{2}$ and $|1_L\rangle$ is encoded as $(|000\rangle - |111\rangle)^{\otimes 3}/2\sqrt{2}$, which protects against an arbitrary error on a single qubit.

3.1 Physical error model

Several properties of quantum error-correction codes are functions of the underlying physical error model. In this work, we assume incoherent circuit-level noise for the physical qubits, meaning that each physical gate, state initialisation, and measurement outcome is affected by a random Pauli error with probability p_{phy}. More precisely, at any point of a quantum circuit, the quantum state of a physical qubit is mapped according to

$$\rho \mapsto (1 - p_{\text{phy}})\rho + \frac{p_{\text{phy}}}{3} \cdot \mathsf{X}\rho\mathsf{X} + \frac{p_{\text{phy}}}{3} \cdot \mathsf{Y}\rho\mathsf{Y} + \frac{p_{\text{phy}}}{3} \cdot \mathsf{Z}\rho\mathsf{Z}.$$

Even though two-qubit gates are more prone to errors than single-qubit gates, we consider a single characteristic error rate p_{phys} for both types of gate in circuit-level noise. We will assume that $p_{\text{phy}} = 10^{-5}$ throughout, which is an optimistic but not unrealistic assumption [29, 80, 146, 59, 157, 178].

3.2 Surface codes

The codes mentioned above are only resilient to very small physical errors. This was later greatly improved with the introduction of surface codes by Kitaev [120, 121]. In surface codes, the physical qubits are arranged in a two-dimensional array on a surface of non-trivial topology, e.g., a plane or a torus, and quantum operations are associated with non-trivial homology cycles of the surface. Surface codes have several appealing properties, e.g., very high error tolerance [195, 164] and local check (stabiliser) measurements. We will not review the surface code in detail here, but we will quote important properties that will be used in our analysis. For more details on the surface code, see [65, 79, 138, 60].

There are a few different encoding schemes for surface codes, e.g., defect-based [79], twist-based [40], and patch-based [102] encodings. Here we shall work exclusively with the latter, since surface-code patches offer lower space overhead and low-overhead Clifford gates [49, 143]. A (rotated) surface-code patch of distance d employs d^2 physical qubits to encode one logical qubit and is able to correct arbitrary errors on any $\lfloor (d-1)/2 \rfloor$ qubits. In order to extract information from a surface code and check for errors, its check operators are measured, which requires d^2 extra measurement qubits for a total of $2d^2$ physical qubits. Moreover, the subroutine of measuring check operators naturally sets a time scale in any experiment. By a *code cycle* we mean the time required to measure all surface-code check operators. It is also common to define a *logical cycle* as d code cycles [138], since $\Omega(d)$ check-operator measurements are required to successfully discern measurement errors from physical errors. We will assume that a quantum computer can perform one code cycle every 100 ns, which is quite an optimistic but not unrealistic assumption [58, 179, 126, 30].

One of the main results of the theory of quantum fault tolerance is the *threshold theorem* [123, 7, 120, 124, 170, 93, 161]. On a high level, it states that, under some reasonable assumptions about the noise of the underlying hardware, an arbitrary long quantum computation can be carried out with arbitrarily high reliability, provided the error rate p_{phy} per quantum gate is below a certain critical *threshold* value p_{th}. Applied to the surface code specifically, the threshold theorem states that the probability of a *logical error* occurring on a distance-d surface code after measuring the check operators and correcting for the observed physical errors vanishes exponentially with the distance d as long as p_{phy} is below the threshold p_{th}. This means that a quantum computation can be made arbitrarily reliant by increasing the distance of the surface-code patch. The surface code exhibits a very high threshold [195, 164, 188] for most error models, and has a threshold of approximately 1% for circuit-level noise [196, 191]. Therefore, under a circuit-level noise model with error p_{phy}, the logical error rate per logical qubit per code cycle can be approximated as [78]

$$p_L(p_{\text{phy}}, d) = 0.1(100 p_{\text{phy}})^{(d+1)/2}.$$

If we wish that n logical qubits survive for T code cycles with high probability, say 99.9%, then the probability that a logical error affects any logical qubit during all code cycles must be smaller than 0.1%. This determines the required code distance d when encoding the logical qubits as

$$T \cdot n \cdot p_L(p_{\text{phy}}, d) < 0.001.$$

3.3 Baseline architecture vs active-volume architecture

It is necessary to specify a physical architecture for a general-purpose fault-tolerant quantum computer which, together with a compilation scheme, converts quantum computations into instructions for that architecture. There are mainly two types of architectures that will be taken into consideration in this work: baseline architectures with nearest-neighbor logical two-qubit interactions on a 2D grid [138, 78, 55, 56, 41], and the active-volume architecture [142] that employs a logarithmic number of non-local connections between logical qubits.

In baseline architectures, the most relevant parameters are the number of data qubits n_Q (i.e., the number of logical qubits on a circuit-level quantum computation) and the number of non-Clifford gates, which in our case is the number of Toffoli gates n_{Toff}. This is because all Clifford gates can be commuted to the end of the computation and be absorbed by final measurements [138]. Both quantities n_Q and n_{Toff} define the *circuit volume* $n_Q \cdot n_{\text{Toff}}$, which is proportional to the *spacetime volume cost* of the quantum computation, i.e., the total number of logical qubits taking into consideration space overheads multiplied by the total number of logical cycles. In baseline architectures, a n_Q-qubit quantum computation consists roughly of $2n_Q$ logical qubits. To be more precise, using Litinksi's fast data blocks [138], an n_Q-qubit quantum computation requires $2n_Q + \sqrt{8n_Q} + 1$ logical qubits in total (in order to efficiently consume magic states). On the other hand, one Toffoli gate is executed in 6 logical cycles, or in 4 logical cycles if the target qubit is in the $|0\rangle$ state, which will be the case of almost all Toffoli gates in our circuits.

The figure of merit in baseline architectures is the circuit volume. Most of the time, however, a large portion of the circuit volume is *idle volume*, i.e., volume attributed to qubits that are not part of an operation at a certain time and are thus idling. Since idling qubits have the same cost as active qubits when using surface codes, the cost of logical operations scales with the number of logical qubits n_Q. In active-volume architectures, on the other hand, only active qubits contribute to the spacetime volume cost of a quantum computation. More specifically, in active-volume architectures, a quantum computer is made up of modules with d^2 physical qubits. Each module can operate as memory or a workspace module. A memory module increases the memory capacity by one logical qubit, and a workspace module increases the computational speed by one *logical block* per logical cycle. An operation is measured in terms of logical blocks and its cost is basically the amount of workspace modules it requires per logical cycle. We assume that n_Q logical qubits result in $n_Q/2$ memory qubits and a speed of $n_Q/2$ logical blocks per logical cycle. The figure of merit in an active-volume architecture is the number of logical blocks, called *active volume*. In order to obtain the total active volume of a quantum computation, we must simply sum up all the active volume of its constituent operations, several of which were given in [142], e.g., a Toffoli has an active volume of 12 plus the active volume of distilling a magic state (see Section 3.4). Litinski and Nickerson [142] proposed a general-purpose active-volume architecture that executes quantum computations with spacetime volume cost of roughly twice the active volume. Contrary to baseline architectures, active-volume ones rely on non-local connections between components, which allows for several fast operations. As an example, Bell measurements can be performed in one code cycle, while in baseline architectures it requires 2 logical cycles via lattice surgery [102, 143, 78]. We point the reader to [142] for a detailed list of assumptions.

Common to both architectures is the time required to perform a layer of single-qubit measurements (or Bell measurements in active-volume architecture), feed the measurement outcomes into a classical decoder, perform a classical decoding algorithm like minimum-weight perfect matching [74, 65] or union-find [64, 63], and use the result to send new instructions to the quantum computer, which is called *reaction time* τ_r. In this work, we shall assume a reaction time of 1 μs, which is an optimistic assumption, as most previous works assume a reaction time of 10 μs [86, 140]. Related to the reaction time is the *reaction depth* of a quantum computation, which is the number of reaction layers, i.e., layers of reactive measurements that must be classically decoded and fed back into the circuit. We thus distinguish between the time required to execute all gates in a circuit when the reaction time is zero (*circuit time*) and the reaction depth times the reaction time (*circuit reaction (time) limit*).

3.4 Magic state distillation

It is known that no quantum error-correction code can transversally implement a universal gate set [73], i.e., be physically implemented on a logical qubit by independent actions of single-qubit physical gates on a subset of the physical qubits. For surface codes, this means T and CCZ gates, among others. In order to overcome this problem, a resource state is first prepared separately and subsequentially consumed to execute a non-transversal gate like a T or a CCZ gate [48]. For T gates, the resource state is a magic state $|T\rangle = (|0\rangle + e^{i\pi/4}|1\rangle)/\sqrt{2}$, while for CCZ gates the resource state is $|CCZ\rangle = \mathsf{CCZ}|+\rangle^{\otimes 3}$. A magic state $|T\rangle$ can be used to perform a T gate by measuring the logical Pauli product $\mathsf{Z} \otimes \mathsf{Z}$ acting on an input state and the magic state [144, 138, 139] akin to teleportation protocols (cf. [161, Figure 10.25]). A similar procedure can be used to perform a CCZ gate by consuming one $|CCZ\rangle$ state (see [142, Figure 14(a)]). However, applying a physical T or CCZ gate onto a few physical qubits yields a resource state with physical error p_{phy}. If this resource state is then used to perform a logical gate, the error rate of the logical gate will be proportional to p_{phy}, which can be too high for a long computation and will spoil the final outcome. One common procedure to generate low-error magic states is to employ magic state distillation protocols [48].

Magic state distillation is a short error-detecting quantum procedure to generate a low-error magic state from several high-error magic state copies. First introduced by Bravyi and Kitaev [48] and Reichardt [176], several different protocols have since been developed [47, 77, 150, 109, 72, 71, 52, 163, 97, 53, 138, 139]. There are a few different but equivalent ways to understand magic state distillation. One is to create the logical magic state using an error-correction code with transversal T gates, e.g., punctured Reed-Muller codes [48, 97] or code-blocks [47, 109, 77]. As an example, the 15-to-1 distillation procedure [48, 176, 78] employs a punctured Reed-Muller code to first encode a logical $|+_L\rangle$ state within 15 physical qubits. The transversallity of the code allows to perform a logical T_L gate onto $|+_L\rangle$ from individual physical T gates, which yields $\mathsf{T}_L|+_L\rangle$. The encoding procedure is then uncomputed and the logical information is shifted to one of the physical qubits. Measuring the remaining physical qubits gives information on possible errors and on whether the procedure was successful or not. If the error probability of the 15 T gates is p_{phy}, then the error probability of the output state is $35p_{\text{phy}}^3$, where the factor 35 comes from different error configurations that are not detectable by the protocol. As a result, 15 magic states with error p_{phy} are distilled down to one magic state with error $35p_{\text{phy}}^3$. A similar distillation procedure exists for creating a low-error $|CCZ\rangle$ state, e.g., Gidney and Fowler [87] proposed an 8-to-CCZ distillation protocol to output a $|CCZ\rangle$ state with error $28p_{\text{phy}}^2$ from 8 $|T\rangle$ states with error p_{phy}. In order to achieve lower error rates than $35p_{\text{phy}}^3$ or $28p_{\text{phy}}^2$, it is possible to concatenate different distillation protocols, meaning that the output states of a level-1 distillation protocol can serve as input magic states for a level-2 distillation protocol, etc.

For baseline architectures, we shall employ the magic state distillation protocols from Litinski [139] which are, as far as we are aware, one of the best to this day. Litinski's protocols are characterised by three code distances d_X, d_Z, d_m from several internal patches. As shown in [139], a $(15\text{-to-}1)_{d_X,d_Z,d_m}$ distillation protocol outputs a low-error magic state every $6d_m$ code cycles using $2(d_X+4d_Z)\cdot 3d_X+4d_m$ physical qubits. Similarly, a two-level protocol is described by three additional code distances d_{X2}, d_{Z2}, and d_{m2}, plus the number n_{L1} of level-1 distillation blocks, where n_{L1} is an even integer. As an example quoted from Litinski's paper [139, Table 1], if $p_{\text{phy}} = 10^{-4}$, then the $(15\text{-to-}1)^4_{7,3,3} \times (8\text{-to-CCZ})_{15,7,9}$ protocol outputs a $|CCZ\rangle$ state with error $p_{\text{out}} = 7.2\cdot 10^{-14}$ in 36.1 code cycles using $12,400$ physical qubits. As will be clear in Section 8, we shall require higher-than-two-level protocols to achieve error rates below 10^{-40}. Even though Litinski [139] only focuses on one and two-level distillation protocols, it is not hard to continue with their analysis and derive the resources required for a three-level distillation protocol: we simply input level-2 magic states into a level-3 protocol with code parameters d_{X3}, d_{Z3}, and d_{m3}, plus the number n_{L2} of level-2 distillation blocks. When optimising the code distances, one usually finds that $d_X = d$, $d_Z \approx d/2$, $d_m \approx d/2$ [139, 142]. We shall then consider concatenated protocols of the form $(15\text{-to-}1)^{n_{L1}}_{d/4,d/8,d/8} \times (15\text{-to-}1)^{n_{L2}}_{d/2,d/4,d/4} \times (8\text{-to-CCZ})_{d,d/2,d/2}$.

Regarding active-volume architectures, on the other hand, we employ the distillation protocols from [142] of the form $(15\text{-to-}1)_{d,d/2,d/2}$ and $(8\text{-to-CCZ})_{d,d,d/2}$. Given a quantum computation with logical blocks of distance d, then a $(15\text{-to-}1)_{ad,ad/2,ad/2}$ protocol has an active volume of $35a^2/2$, while a $(8\text{-to-CCZ})_{ad,ad,ad/2}$ protocol has an active volume of $25a^2/2$. Therefore, a $(15\text{-to-}1)^{n_{L1}}_{d/4,d/8,d/8} \times (15\text{-to-}1)^{n_{L2}}_{d/2,d/4,d/4} \times (8\text{-to-CCZ})_{d,d,d/2}$ protocol has an active volume of $\frac{35}{32}n_{L_1}n_{L_2} + \frac{35}{8}n_{L_2} + \frac{25}{2}$. By using $n_{L_1} = 8$ level-1 protocols and $n_{L_2} = 4$ level-2 protocols, 16 level-1 distilled $|T\rangle$ states are produced every $d/4$ code cycles, and 8 level-2 distilled $|T\rangle$ states are produced every $d/2$ code cycles, meaning that one level-3 distilled $|CCZ\rangle$ can be produced every d code cycles. Therefore, the $(15\text{-to-}1)^8_{d/4,d/8,d/8} \times (15\text{-to-}1)^4_{d/2,d/4,d/4} \times (8\text{-to-CCZ})_{d,d,d/2}$ protocol has an active volume of 65 and produces a $|CCZ\rangle$ state every logical cycle. The output error can be calculated using the approximate expressions in [142], or using the Python file for the baseline-architecture distillation protocols from [139].

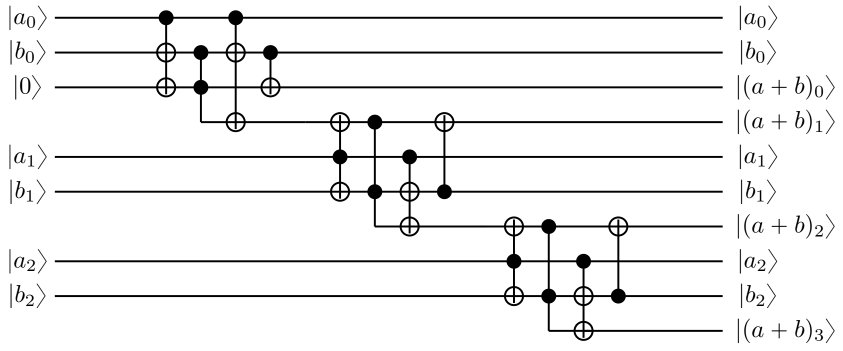


Figure 2: Gidney's out-of-place quantum adder (modulo 2^κ) that adds two κ-bit numbers a and b stored in quantum registers.

4 Arithmetic on a quantum computer

In this section, we turn our attention to the resources needed to perform some simple arithmetic operations on a quantum computer that will be the building blocks for the analysis of quantum sieving. But first, we need a way to store D-dimensional vectors with integer entries in a quantum computer. In order to do that, we store the *two's-complement* representation $x_{\kappa-1}\dots x_0$ of a κ-bit integer x in a κ-qubit quantum register $|x_{\kappa-1},x_{\kappa-2},\dots,x_0\rangle$, where $x_0,\dots x_{\kappa-1}\in\{0,1\}$. We do not use a sign-magnitude representation for an integer $x=(-1)^{x_{\kappa-1}}(x_{\kappa-2}\cdot 2^{\kappa-2}+\cdots+x_0\cdot 2^0)$, as done by other works [200, 68], since addition is non trivial in such representation and several known quantum adders would have to be modified to take negative numbers into consideration. The value of κ is chosen in advance and remains the same throughout the whole computation. Increasing the value of κ of course requires more physical resources for the algorithm execution but at the same time reduces the chance of an overflow occurring. Throughout this work, we assume $\kappa=32$, which translates to a capacity of working with integers in the range $[-2^{31},2^{31}-1]$. To store an entire D-dimensional vector, we store each of its entries separately using the above encoding, so that $D\kappa$ qubits are required in total.

We now start with reviewing fundamental arithmetic operations on a quantum computer: addition, comparison, and multiplication.

4.1 Quantum adders

An *out-of-place quantum adder* (modulo 2^κ) is a unitary that adds two κ-bit integers $x=x_{\kappa-1}\dots x_0$ and $y=y_{\kappa-1}\dots y_0$ together onto a third register,

$$|x_{\kappa-1},\dots,x_0\rangle|y_{\kappa-1},\dots,y_0\rangle|0\rangle^{\otimes\kappa}\mapsto|x_{\kappa-1},\dots,x_0\rangle|y_{\kappa-1},\dots,y_0\rangle|(x+y)_{\kappa-1},\dots,(x+y)_0\rangle.$$

It is possible to define an *in-place quantum adder* which replaces one of the inputs with the outcome, but in this work we shall focus on out-of-place adders since they have a lower Toffoli-count [85].

Several quantum adders or related circuits have been proposed in the past few decades [35, 91, 194, 201, 70, 62, 69, 137, 107, 19, 108, 85, 159, 134, 133], see [166] for a review. As far as we are aware, the state-of-the-art quantum adder in terms of Toffoli-count is due to Gidney [85], which is an improved version of Cuccaro's adder [62]. Gidney's adder (Figure 2) concatenates several copies of the adder building-block, each of which is made of one Toffoli computation and its uncomputation requiring no Toffoli gates. In order to add two κ-bit integers, Gidney's adder requires $\kappa-1$ Toffoli gates in total. Even though Gidney's results are phrased in terms of T gates, we translate them into Toffoli gates. The Toffoli-count, together with several other quantities like Toffoli-width (maximum number of Toffoli gates in a single layer), reaction depth, number of logical qubits are shown in Table 1. Its active volume, on the other hand, was computed by Litinski and Nickerson [142, Table 1] and equals to $(\kappa-1)(39+C_{|CCZ\rangle})+7$, where $C_{|CCZ\rangle}$ is the active volume of distilling one $|CCZ\rangle$ state.

Using Gidney's quantum (out-of-place) adder, it is easy to develop a quantum controlled (out-of-place) adder (modulo 2^κ): first apply the vanilla adder to get $|c\rangle|x\rangle|y\rangle|0\rangle^{\otimes 2\kappa}\mapsto|c\rangle|x\rangle|y\rangle|x+y\rangle|0\rangle^{\otimes\kappa}$, followed by κ Toffoli gates to copy each bit $(a+b)_i$ onto another register controlled on $c \in \{0,1\}$. This yields $|c\rangle|a\rangle|x\rangle|x+y\rangle|c(x+y)\rangle$. It is possible to uncompute the ancillary register $|x+y\rangle$ by performing the inverse of the first adder, which uses no Toffoli gates. However, if we keep such ancillary register, uncomputing the whole controlled adder requires no Toffoli gates, as opposed to $2\kappa + O(1)$ if you call the inverse of the entire controlled adder. Therefore, we shall keep the ancillary register $|x+y\rangle$ until the uncomputation of the whole circuit. Finally, we note that the controlled copying of the ancillary register $|x+y\rangle$ can be done while the out-of-place adder is being performed. The active volume of the whole computation, while not considered by [142], can be easily calculated from the its separated parts. The results are described in Table 1.

4.2 Quantum comparator

A quantum comparator is a unitary that compares whether a κ-bit integer $x = x_{\kappa-1}\dots x_0$ is bigger than another κ-bit integer $y = y_{\kappa-1}\dots y_0$,

$$|x\rangle|y\rangle|0\rangle \mapsto |x\rangle|y\rangle|\mathbf{1}[x>y]\rangle.$$

A comparator can be obtained from the highest-order bit of the difference $x-y$. Whether we use one's-complement or two's-complement arithmetic, the identity $x - y = \overline{\overline{x} + y}$ holds. Therefore, it is possible to use an out-of-place adder as a comparator: complement x, employ a quantum adder and keep the highest-order bit, and complement the obtained highest-order bit. All the adders described in the previous section are modulo 2^κ, meaning that the highest-order bit is not calculated. Nonetheless, we shall assume that there is no overflow and therefore the highest-order bit of the summation modulo 2^κ yields the correct answer. Moreover, if one of the inputs is classical, say y, then there is no need to complement the quantum register holding x, except maybe for the highest-order bit of $x-y$ depending on whether we are checking for $x>y$ or $x<y$.

4.3 Quantum multipliers

Similarly to addition, we can define an *out-of-place quantum multiplier* (modulo 2^κ) as the unitary that multiplies two κ-bit integers $x = x_{\kappa-1}\dots x_0$ and $y = y_{\kappa-1}\dots y_0$ together and places the outcome on a third register,

$$|x_{\kappa-1},\dots,x_0\rangle|y_{\kappa-1},\dots,y_0\rangle|0\rangle^{\otimes\kappa} \mapsto |x_{\kappa-1},\dots,x_0\rangle|y_{\kappa-1},\dots,y_0\rangle|(x\cdot y)_{\kappa-1},\dots,(x\cdot y)_0\rangle.$$

Several quantum multipliers have been proposed in the past decade [137, 107, 26, 125, 177, 159, 169, 134, 81, 133, 165, 148]. In terms of T-count, the works of Li et al. [133] and Orts et al. [165] are the best as far as we are aware. Li et al. [133] proposed a quantum multiplier with $16\kappa^2 - 14\kappa$ T gates, $\kappa+1$ ancillae, and T-depth of $4\kappa^2+4\kappa+4$. Orts et al. [165], on the other hand, proposed a quantum multiplier with $18\kappa^2 - 24\kappa$ T gates, $2\kappa^2 - 2\kappa + 2$ ancillae, and T-depth of $14\kappa - 14$. Both T-counts are comparable, while the trade-off is between ancillae and T-depth.

Since we are mostly concerned with Toffoli-count and are willing to use extra ancillae (including keeping dirty ones for subsequent uncomputation), we employ a quantum multiplier based on schoolbook multiplication with $\kappa - 1$ out-of-place additions from Table 1. We note that a similar idea appeared before in [180], although not modulo 2^κ, and only very recently, by the time this manuscript was finalised, a similar construction with a similar Toffoli-count was proposed by Litinski [141].

The multiplier works as follows. The input registers $|x_{\kappa-1},\dots,x_0\rangle$ and $|y_{\kappa-1},\dots,y_0\rangle$ are first copied $\kappa-1$ times: the bits x_i and y_i are copied $\kappa-1-i$ times, $i = 0,\dots,\kappa-2$. This can be done with $\kappa^2-\kappa$ CNOTs in depth $\lceil\log_2\kappa\rceil$. We do not need to copy x_i and y_i a number of $\kappa-1$ times since the multiplication is done modulo 2^κ and high-order bits are ignored. We then perform κ steps in parallel: in the i-th step, $i = 0,\dots,\kappa-1$, the qubits $|x_i,\dots,x_0\rangle$ are copied onto fresh ancillae $|0\rangle^{\otimes(i+1)}$ controlled on one copy of $y_{\kappa-1-i}$ using Toffoli gates. At the end of this process, we have κ registers holding all partial sums: the i-th one made up of $i+1$ bits, $i = 0,\dots,\kappa-1$. Then, the κ partial

Table 1: State-of-the-art constructions for several quantum arithmetic circuits on κ-bit integers. All operations are out-of-place, modulo 2^κ, and already include their inverses. The resources are broken down into Toffoli-count, Toffoli-width, reaction depth, qubit-width (ancillae plus input/output qubits), and active volume. Here $C_{|CCZ\rangle}$ is the active volume of distilling one $|CCZ\rangle$ state.

Circuit / Resource	Toffoli-count	Toffoli-width	Reaction depth	Qubit-width	Active volume
Adder/Comparator	$\kappa - 1$	1	$2(\kappa-1)$	3κ	$(\kappa-1)(39 + C_{|CCZ\rangle}) + 7$
Controlled adder	$2\kappa - 1$	κ	2κ	$4\kappa+1$	$(\kappa-1)(51 + C_{|CCZ\rangle}) + 19$
Multiplier	$\kappa^2 - \kappa + 1$	$0.5\kappa^2 + 0.5\kappa$	$2\kappa\log_2\kappa - 2\kappa - 2\log_2\kappa + 4$	$2\kappa^2 + \kappa$	$28\kappa^2 - 42\kappa + 28$ $+(\kappa^2-\kappa+1)C_{|CCZ\rangle}$
Multiplier (hybrid)	$0.5\kappa^2 - 1.5\kappa + 1$	0.5κ	$2\kappa\log_2\kappa - 2\kappa - 2\log_2\kappa + 2$	$1.5\kappa^2 + 0.5\kappa$	$20.25\kappa^2 - 48.75\kappa + 32$ $+(0.5\kappa^2 - 1.5\kappa + 1)C_{|CCZ\rangle}$

sums are added up using out-of-places adders until the final sum is computed. This can be done in any particular order, the amount of resources is left unchanged except for the reaction depth. The optimal reaction depth combination is tree-wise in $\lceil\log_2\kappa\rceil$ layers. For simplicity of analysis, let us assume the combination is done sequentially. More precisely, at layer $i = 1,\dots,\kappa-1$, the sum of the previous layer, which has i bits, is added onto the partial sum with $i+1$ bits, which requires an i-bit out-of-place adder (the least significant digit of the second register is just attached to the result register to form the $(i+1)$-bit answer). This means that the total Toffoli-count (already taking into account the $\sum_{i=0}^{\kappa-1}(i+1) = (\kappa^2+\kappa)/2$ Toffoli gates from the controlled copying) is

$$\frac{\kappa^2+\kappa}{2} + \sum_{i=1}^{\kappa-1}(i-1) = \kappa^2 - \kappa + 1.$$

By keeping all dirty ancillae from the computation, the inverse circuit can be implemented with no Toffoli gates! Regarding ancillae, the initial copying requires $\kappa^2-\kappa$ ancillae, while the controlled copying requires another $(\kappa^2+\kappa)/2$ ancillae. The $\kappa-1$ out-of-place adders require $\sum_{i=1}^{\kappa-1} i = (\kappa^2-\kappa)/2$ ancillae, κ of which will be the output. There are thus $2\kappa^2 - 2\kappa$ dirty ancillae, and the total width is $2\kappa^2+\kappa$ (ancillae plus 3κ input and output qubits).

The active-volume calculation is similar to the Toffoli-count. The $2\kappa^2 - 2\kappa$ CNOTs (taking into consideration the inverse) have an active volume of $2\sum_{i=0}^{\kappa-2}(\frac{3}{2}(\kappa-1-i)+2) = 1.5\kappa^2 + 2.5\kappa - 4$, while the $(\kappa^2+\kappa)/2$ Toffoli gates from the controlled copying (plus inverse) have an active volume of $(14 + C_{|CCZ\rangle})(\kappa^2+\kappa)/2$. Finally, the active volume of all adders is $\sum_{i=1}^{\kappa-1}((i-1)(39+C_{|CCZ\rangle})+7) = 19.5\kappa^2 - 51.5\kappa + 32 + (0.5\kappa^2 - 1.5\kappa + 1)C_{|CCZ\rangle}$. Summing everything up yields the active volume of $28\kappa^2 - 42\kappa + 28 + (\kappa^2-\kappa+1)C_{|CCZ\rangle}$.

Concerning the reaction depth, assume for simplicity that κ is a power of 2. The controlled copying has reaction depth of 2. On the other hand, the $\kappa-1$ out-of-place adders are distributed in $\log_2\kappa$ layers, and at the j-th layer, $j = 0,\dots,\log_2\kappa - 1$, we sort the partial sums in increasing number of bits and add up the i-th partial sum with the $(\kappa/2^j - i + 1)$-th partial sum, $i = 1,\dots,\kappa/2^{j+1}$. For example, at the 0-th layer, the partial sum with i bits is added to the partial sum with $\kappa - i + 1$ bits, which requires an i-bit quantum adder, $i = 1,\dots,\kappa/2$ (the $\kappa - 2i + 1$ least significant bits of the larger partial sum are simply attached to the result register to form the κ-bit answer). At the j-th layer, there are $\kappa/2^j$ partial sums, ranging from $1 + (1-2^{-j})\kappa$ bits to κ bits. The longest addition at the j-th layer is the summation between the partial sums with $(1-2^{-j-1})\kappa$ and $1 + (1-2^{-j-1})\kappa$ bits, which requires an $(1-2^{-j-1})\kappa$-bit quantum adder with reaction depth of $2(1-2^{-j-1})\kappa - 2$. Therefore, the total reaction depth is

$$2 + 2\sum_{j=0}^{\log_2\kappa - 1}((1-2^{-j-1})\kappa - 1) = 2\kappa\log_2\kappa - 2\kappa - 2\log_2\kappa + 4.$$

Hybrid classical-quantum inputs. In the case when one of the inputs is classical, say y, then the amount of resources decrease a bit. This is because there is no need to copy the registers $|x\rangle$ and $|y\rangle$ a number of $\kappa - 1$ times at the beginning, since $|y\rangle$ becomes classical and there is no need to parallelise CNOT gates. Moreover, the $\mathsf{Toffoli}$ gates used to controlled copy the register $|x_i, \dots, x_0\rangle$ using $y_{\kappa-1-i}$ become classically controlled CNOT gates. This means that the $\mathsf{Toffoli}$-count decreases by $(\kappa^2 + \kappa)/2$, while the CNOT-count decreases to $\kappa^2 + \kappa$ (already taking the inverse into consideration), which have an active volume of $\sum_{i=0}^{\kappa-1}(\frac{3}{2}(\kappa - i) + 2) = 0.75\kappa^2 + 2.75\kappa$.

5 Grover's quantum search algorithm

Unstructured search can be defined as follows. Given a function $f : \{0,1\}^n \to \{0,1\}$, find a marked element $x \in \{0,1\}^n$ such that $f(x) = 1$, or determine that with high probability, no such input exists. Classically this requires $\Theta(2^n)$ evaluations of f to find such an input or determine it does not exist with high probability. By contrast, Grover [95, 96] designed a quantum algorithm that finds a marked element with high probability and requires only $O(2^{n/2})$ calls to an binary oracle $\mathcal{U}_f$ that evaluates f in superposition, $\mathcal{U}_f : |x\rangle|y\rangle \to |x\rangle|y \oplus f(x)\rangle$ for all $x \in \{0,1\}^n$ and $y \in \{0,1\}$. It was later shown [36, 43, 203, 31] that Grover's algorithm is optimal for unstructured search.

Grover's algorithm is depicted in Figure 3. By starting with the state $2^{-n/2}\sum_{x\in\{0,1\}^n}|x\rangle$ (which can be obtained from $|0\rangle^{\otimes n}$ by applying one layer $\mathsf{H}^{\otimes n}$ of n Hadamard gates), the algorithm repeatedly applies the so-called *Grover operator*

$$\mathsf{G} = \mathsf{H}^{\otimes n}(2|0^n\rangle\langle 0^n| - \mathsf{I}_{2^n})\mathsf{H}^{\otimes n}\mathcal{O}_f$$

and then measures the state on the computational basis. The operator $\mathsf{D} := \mathsf{H}^{\otimes n}(2|0^n\rangle\langle 0^n| - \mathsf{I}_{2^n})\mathsf{H}^{\otimes n}$ is called *diffusion operator* and performs a conditional phase shift such that $|x\rangle \mapsto (-1)^{\mathbf{1}[x\neq 0^n]}|x\rangle$ for all $x \in \{0,1\}^n$. The oracle $\mathcal{O}_f$, on the other hand, is defined as $\mathcal{O}_f : |x\rangle \mapsto (-1)^{f(x)}|x\rangle$ for all $x \in \{0,1\}^n$. We note that it is possible to implement the phase oracle $\mathcal{O}_f$ from the binary operator $\mathcal{U}_f$ by simply applying $\mathcal{U}_f$ onto $|x\rangle|-\rangle$, where $|-\rangle := (|0\rangle - |1\rangle)/\sqrt{2}$.

Let $N = 2^n$ be the number of elements and M the number of marked elements such that $f(x) = 1$. It can be shown [95, 96, 46] that after m iterations of G, the probability of measuring a marked element is $\sin^2((2m+1)\theta)$, where $\sin^2\theta = \sqrt{M/N}$. Therefore, by using $m = \lfloor\frac{\pi}{4}\sqrt{N/M}\rfloor$ iterations, the measurement outcome will be a marked state with probability at least $1 - \frac{M}{N}$, which is sufficiently close to 1 for $N \gg 1$. Each iteration requires one query to the oracle $\mathcal{O}_f$ (or $\mathcal{U}_f$) and one application of the diffusion operator. The diffusion operator, in turn, requires $2n$ Hadamard gates and one conditional phase $|x\rangle \mapsto (-1)^{\mathbf{1}[x\neq 0^n]}|x\rangle$, which is basically a slightly modified multi-controlled $\mathsf{Toffoli}$. More precisely, $2|0^n\rangle\langle 0^n| - \mathsf{I}_{2^n}$ equals $(\mathsf{X}^{\otimes n}\otimes\mathsf{I})(\mathsf{C}^{(n)}\text{-}\mathsf{X})(\mathsf{X}^{\otimes(n+1)})$ applied onto $|x\rangle|-\rangle$. The multi-controlled gate $\mathsf{C}^{(n)}\text{-}\mathsf{X}$ can be implemented using $n-1$ $\mathsf{Toffoli}$ gates and $n-2$ ancillae according to Fact 4 (among other resources).

We summarise the above discussion in the following result.

Fact 6 (Grover's algorithm)**.** *Let the positive integers $N = 2^n$ and M. Consider a Boolean function $f : \{0,1\}^n \to \{0,1\}$. Assume $M = |\{x \in \{0,1\}^n : f(x) = 1\}|$ is known and we have access to a quantum oracle $\mathcal{O}_f : |x\rangle \mapsto (-1)^{f(x)}|x\rangle$. Then it is possible to find one marked element of f with probability at least $1 - \frac{M}{N}$ by using $\lfloor\frac{\pi}{4}\sqrt{N/M}\rfloor$ queries to $\mathcal{O}_f$ and the diffusion operator D.*

Fact 6 assumes that the number of solutions is known beforehand, which is usually not the case. Nonetheless, there is a variant of Grover's algorithm due to Boyer, Brassard, Høyer, and Tapp [43] (see also [202]) that applies to the case when the number of solutions is not known ahead of time. The main idea of their algorithm is to start with some parameter m, choose an integer j uniformly at random such that $0 \leq j < m$, and perform j iterations of Grover's search. If it does not return a solution, the value m is increased to λm for any constant $1 < \lambda < 4/3$ and the procedure is repeated. Boyer et al. [43] showed that this algorithm finds a solution (or determines that no solution exists) with high

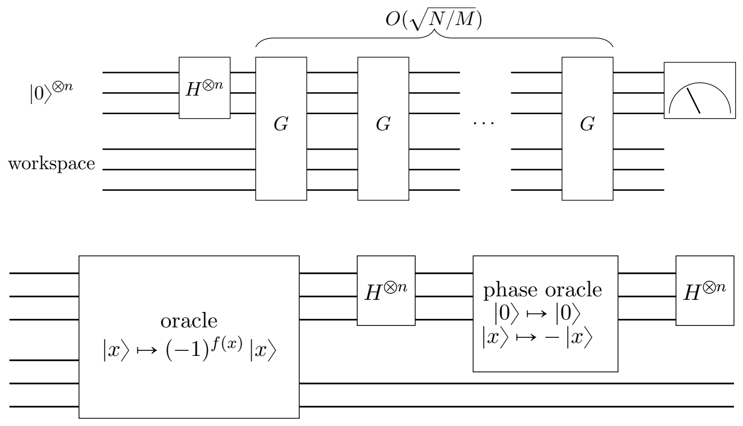


Figure 3: Circuit for Grover's search algorithm (top) and the Grover oracle G (bottom).

probability in expected time $O(\sqrt{N/M})$. A very thoroughly analysis of Grover's algorithm has been done by Cade, Folkertsma, Niesen, and Weggemans [50], which we quote next and expand with the necessary resources for the diffusion operator.

Fact 7 ([50, Lemma 4])**.** *Let $\delta \in (0,1)$ and the positive integer $N = 2^n$. Consider a Boolean function $f : \{0,1\}^n \to \{0,1\}$ with $|\{x \in \{0,1\}^n : f(x) = 1\}| = M$, where $0 \le M < N/4$ and the value M is not necessarily known. Assume we have access to a quantum oracle $\mathcal{O}_f : |x\rangle \mapsto (-1)^{f(x)}|x\rangle$ and let $\mathsf{D} := \mathsf{H}^{\otimes n}(2|0^n\rangle\langle 0^n| - \mathsf{I}_{2^n})\mathsf{H}^{\otimes n}$ be the diffusion operator. Then there is a quantum algorithm that, with probability at least $1-\delta$,*

- *returns $x \in \{0,1\}^n$ such that $f(x) = 1$ if such a solution exists by using an expected number $\lceil 7.67\sqrt{N/M}\rceil$ of queries to $\mathcal{O}_f$ and D,*
- *or concludes that no such solution exists by using $\lceil 9.2\sqrt{N}\log_3(1/\delta)\rceil$ queries to $\mathcal{O}_f$ and D.*

Moreover, one call to the diffusion operator requires $n-1$ Toffoli gates and $n-1$ ancillae, and has reaction depth of $2\lceil\log_2 n\rceil$, Toffoli-width of $\lfloor n/2\rfloor$, and active volume of $(n-1)(18 + C_{|CCZ\rangle})$, where $C_{|CCZ\rangle}$ is the active volume of distilling one $|CCZ\rangle$ state.

We mention that there exists an exact version of Grover's algorithm that succeeds with probability 1 (see [46, Section 2.1] and [199, Exercise 7.5]). However, even though this version is query efficient, it is not necessarily gate efficient, therefore we will not use it.

6 Quantum random access memory (QRAM)

A quantum random access memory (QRAM) is the quantum analogue of the classical random access memory (RAM) device which allows access to classical or quantum data in superposition. From a architecture perspective, a QRAM of size 2^n and precision κ is composed of a $\kappa 2^n$-(qu)bit memory register that stores either κ bits or qubits in each of 2^n different cells, an n-qubit address register which points to the memory cell to be addressed, a κ-qubit target address into which the content of the addressed memory cell is copied, and an $O(2^n)$-qubit auxiliary register that intermediates the copying of the memory register into the target register controlled on the address register. For more details on the architecture of QRAMs, we point the reader to [98, 167, 106, 16].

In general, a QRAM allows access to either classical or quantum data stored in some register. Throughout this paper, we shall work exclusively with QRAMs that access *classical* data, which are sometimes referred to as quantum random access classical memory (C-QRAM or QRACM). For simplicity, we stick to the usual QRAM nomenclature. Moreover, we shall consider QRAM calls that keep a garbage register (dirty ancillae) in order to aid their uncomputation at latter stages.

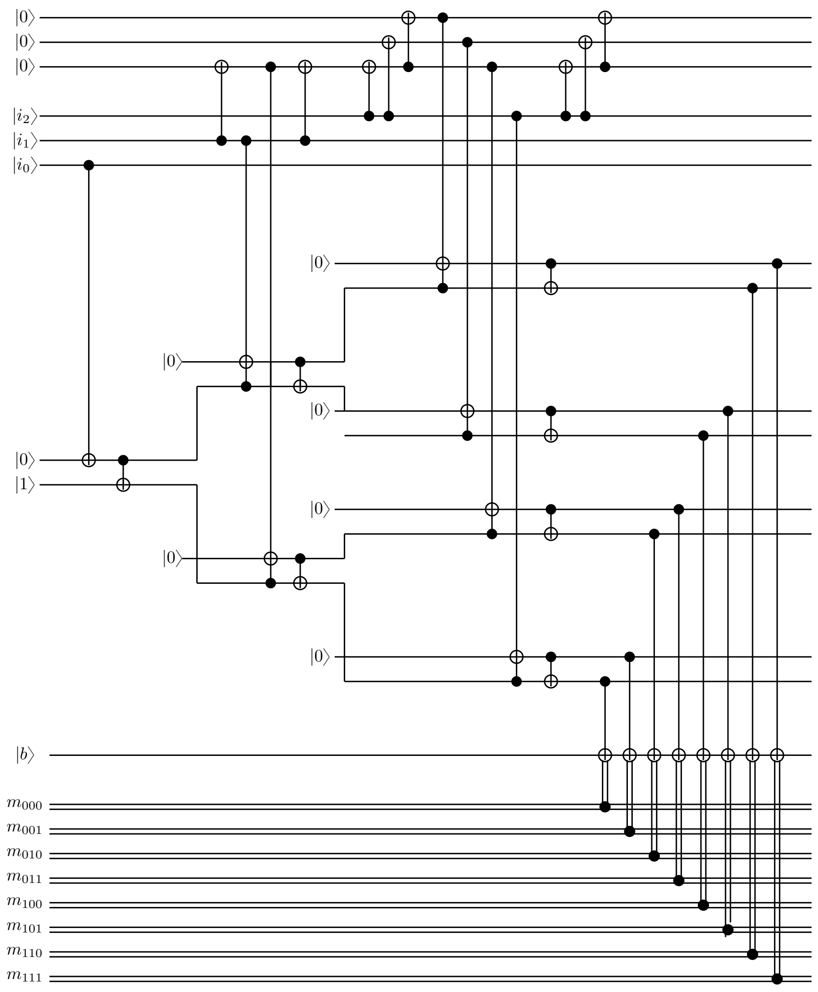


Figure 4: The bucket-brigade QRAM circuit from Arunachalam et al. [23]. In every layer, before the parallel layer of Toffoli gates, a log-depth linear-size gadget copy the index register so the Toffoli gates can be executed in parallel.

Definition 8 (Quantum random access memory (QRAM))**.** *A* QRAM *of size* 2^n *and precision* κ *is a device that stores classical, indexed data* $\{x_i \in \{0,1\}^\kappa : i \in [2^n]\}$ *and allows oracle queries*

$$\mathsf{QRAM} : |i\rangle\,|0\rangle^{\otimes\kappa}\,|\bar{0}\rangle \mapsto |i\rangle\,|x_i\rangle\,|\text{garbage}_i\rangle, \quad \forall i \in [2^n].$$

The first architectures for QRAM were proposed and formalised in [90, 89], namely the Fan-Out and bucket-brigade architectures. In these architectures, the memory register is accessed by a binary tree of size $O(2^n)$ and depth n. Each qubit of the address register controls the direction from the root down to the correct memory cell within the binary tree, i.e., the k-th address qubit specifies whether to go left or right at a router on the k-th level of the binary tree. The target is sent down the tree and is routed controlled on the address qubits at each level until the memory register, at which point the information is copied into the target and the target is sent back up the tree. The difference between the Fan-Out and bucket-brigade architectures is in how the target qubits are routed down the binary tree. We point out the reader to [90, 89, 23, 99] for more information.

Here we shall be agnostic regarding the underlying architecture of a QRAM and shall work with the circuit model instead. We assume nonetheless that the contents of the memory are stored statically, meaning that the classical data is stored in an external physical hardware, e.g., a tape, which is quantumly queried. This is accomplished by applying classically controlled CNOT gates onto the target qubit with one classical control (a bit from the memory) and one quantum control (a qubit from the last layer of the binary tree). We show a circuit implementation of QRAM in Figure 4. Moreover, we also assume that the classical memory can be updated independently from the QRAM device itself. In other words, m different cells from the classical memory can be rewritten in time $O(\kappa m)$ without the need to update the remaining registers from the QRAM. This differs from Quantum Read-Only Memory (QROM) or table lookups [25] which usually encode the memory content into the circuit layout.

The fault-tolerance resources required to implement a QRAM have been studied by a few works [66, 145, 142, 140]. Di Matteo, Gheorghiu, and Mosca [66] studied the amount of T gates in bucket-brigade style QRAMs, while Low, Kliuchnikov, and Schaeffer [145] proposed a T-efficient sequential QROM circuit. Litinski and Nickerson [142] worked out the active volume of Low et al. proposal. Here we employ a bucket-brigade QRAM due to its exponentially smaller reaction depth compared to Low, Kliuchnikov, and Schaeffer's QROM.

Lemma 9 (Bucket-brigade QRAM)**.** *One bucket-brigade* QRAM *call of size* 2^n *and precision* κ *requires (already including its uncomputation)* $2^n - 2$ Toffoli *gates,* $2^{n+1} - n - 1$ *dirty ancillae (plus* $n + \kappa$ *input/output qubits), and has* Toffoli*-width of* 2^{n-1}*, reaction depth of* $2(n-1)$*, and active volume of* $(25 + 1.5\kappa + C_{|CCZ\rangle})2^n$*.*

Proof. All the resources apart from the active volume are straightforward. In the following, we already take the uncomputation into consideration. A bucket-brigade QRAM can be divided into $2^n - 2$ blocks made up of one Toffoli and one CNOT gate and having active volume $14 + 2 \cdot 4 + C_{|CCZ\rangle}$; $\kappa \cdot 2^n$ classically controlled CNOTs with average active volume of $(\frac{3}{2}2^n + 1)\kappa$ (since on average half the CNOTs is actually performed); and $2^n - n - 1$ CNOTs to copy the address register with active volume of $2\sum_{i=0}^{n-1}(\frac{3}{2}(2^i - 1) + 2) = 3 \cdot 2^n + n - 3$. Summing all active volumes yields the result after some simple approximations. □

7 The shortest vector problem and sieving algorithms

The most important problem on lattices and that underlies many lattice-based cryptography functions [10, 173, 175, 154] is the *shortest vector problem* (SVP). Given a set $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_N\} \subset \mathbb{R}^D$ of N linearly independent vectors, the set

$$\mathcal{L}(\mathbf{B}) := \left\{ \sum_{j=1}^{N} \lambda_j \mathbf{b}_j : \lambda_1, \dots, \lambda_N \in \mathbb{Z} \right\}$$

of all integer linear combinations of $\mathbf{B}$ is called the lattice associated with $\mathbf{B}$. The set $\mathbf{B}$ is called the basis of the lattice, while the integers N and D are its rank and dimension, respectively. In this work, we consider *full rank* lattices, which is the case when $N = D$. The minimum distance $\lambda(\mathcal{L})$ of a lattice $\mathcal{L}$ is the length of its shortest non-zero lattice vector, $\lambda(\mathcal{L}) := \min\{\|\mathbf{x}\| : \mathbf{x} \in \mathcal{L} \setminus \{\mathbf{0}\}\}$. We shall abuse notation and write $\lambda(\mathbf{B})$ instead of $\lambda(\mathcal{L}(\mathbf{B}))$.

Definition 10 (Shortest vector problem)**.** *Given a lattice basis* $\mathbf{B} \in \mathbb{R}^{D \times D}$*, find* $\mathbf{x} \in \mathcal{L}(\mathbf{B})$ *such that* $\|\mathbf{x}\| = \lambda(\mathbf{B})$*.*

SVP is known to be NP-hard under randomised reductions [9, 151, 152] given an arbitrary basis of an arbitrary lattice. Even the approximate version of SVP, wherein one is tasked to find a lattice vector with norm at most $(1 + \epsilon)\lambda(\mathbf{B})$ for $\epsilon > 0$, is known to be NP-hard [114, 115]. Nonetheless, several exponential-time algorithms have been proposed in the past few decades to tackle SPV. There are currently three main methodologies: enumeration [76, 112, 168], sieving [12, 11, 156, 5], and constructing the Voronoi cell of the lattice [6, 155]. Whereas enumeration has a polynomial space complexity but a superexponential time complexity $O(2^{D \log D})$ on the dimension D of the lattice [168, 182, 183, 112], the remaining methods all have both exponential space and time complexities.

In this section, we focus on and review the major sieving algorithms since their introduction by Ajtai, Kumar, and Sivakumar [12, 11]. Sieving algorithms work by sampling a long list $L = \{\mathbf{v}_1, \dots, \mathbf{v}_m\}$ of lattice vectors (either initially or during the algorithm) and considering all pair-wise differences

$\mathbf{v}_i \pm \mathbf{v}_j \in \mathcal{L}(\mathbf{B})$ from the list. Most of these combinations result into longer vectors than the initial vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, but some lead to shorter vectors. By keeping the resulting shorter vectors into a new list, progress is made into finding the shortest vector. The step of combining lattice vectors from a list in order to form a new list with shorter lattice vectors is called *sieving*. We hope that, if a substantially large number of lattice vectors is sampled, then several sieving steps will result into a small list that contains the shortest vector.

Whereas Ajtai, Kumar, and Sivakumar [12, 11] originally proved that sieving can solve SVP in time and space $2^{\Theta(D)}$, later works improved their results and showed that sieving can provably solve SVP in time $2^{2.465D+o(D)}$ and space $2^{1.233D+o(D)}$ [160, 172, 100]. At first glance, these provable bounds suggest that sieving algorithms would perform poorly in practice, and that solving SVP on dimension beyond 50 would be impractical. Experimental works suggest otherwise and that sieving algorithms perform well in practice. This has led to new sieving proposals that can tackle SVP under *heuristic* assumptions. The first proposal for a heuristic sieving algorithm was given by Nguyen and Vidick [160], which we now review. In what follows, we assume that all the vectors have coordinates described using κ-bits.

7.1 The Nguyen-Vidick sieve

Nguyen and Vidick [160] proposed the first sieving algorithm that relies on heuristic assumptions. A version of the Nguyen-Vidick sieve (`NVSieve`) that already incorporates Grover's algorithm is depicted in Algorithm 1 (cf. [128, Algorithm 2]). The first step is to sample a list L of lattice vectors using, e.g., Klein's algorithm [122, 83], which samples lattice vectors from a distribution that is statistically close to a discrete Gaussian on a lattice with a reasonably small variance. A sieving process is then applied onto L to reduce pairs by considering the differences $\mathbf{v}-\mathbf{w}$ of pairs of lattice vectors $\mathbf{v}, \mathbf{w} \in L$. If $\mathbf{v}-\mathbf{w}$ yields a shorter vector than $\mathbf{v}, \mathbf{w}$, it is stored in a new list L'. Instead of considering all pair-wise combinations of vectors from the list L, the `NVSieve` keeps a list of centers $S \subset L$, each covering a part of the space. Each vector $\mathbf{v} \in L$ from the list is thus combined with vectors $\mathbf{w} \in S$ from the list of centers. If the result is a shorter vector, $\mathbf{v}-\mathbf{w}$ is added to new list L', otherwise the initial list vector $\mathbf{v} \in L$ is added to the list of centers S to cover a part of the space which was previously left uncovered. At the end of the sieving step, $L \leftarrow L'$. After many sieving steps as necessary, the list L contains the shortest lattice vector or is left empty, in which case the whole algorithm is repeated.

Under the heuristic assumption that the angle between two list vectors $\mathbf{v}, \mathbf{w} \in L$ follows the same distribution as the angle between two uniformly random vectors over the unit sphere, Nguyen and Vidick [160] proved that an initial list of size $(4/3)^{D/2+o(D)} = 2^{0.208D+o(D)}$ suffices to find the shortest vector. This bounds the space complexity of the `NVSieve`. On the other hand, the time complexity is dominated by comparing every list vector $\mathbf{v} \in L$ to a center vector $\mathbf{w} \in S$, and since the number of center vectors is asymptotically equivalent to the number of list vectors, this means that the `NVSieve` solves SVP in time $2^{0.415D+o(D)}$.

7.1.1 Numerical experiments and heuristic assumptions

The asymptotic complexity hides a lot of details, especially in case of a quantum algorithm. We want a more refined analysis of the runtime of this algorithm. Since the analysis of the `NVSieve` relies on heuristic assumptions, i.e., that vectors in $L \cap B_D(\gamma, R)$ are uniformly distributed in $B_D(\gamma, R) := \{\mathbf{v} \in \mathbb{R}^D : \gamma R \leq \|\mathbf{v}\| \leq R\}$ (where $R = \max_{\mathbf{v}\in L} \|\mathbf{v}\|$), quantities like the number of sieving steps or the evolution of the list size $|L|$ can behave as random variables and are thus not determined beforehand. Nonetheless, it is possible to assert average trends and worst-case bounds through plausible assumptions and numerical experiments. In the following, we list several observations that shall be useful in forming assumptions.

1. Nguyen and Vidick [160] proved that, given $\gamma \in (2/3, 1)$, the maximum size of the list of centers S is upper-bounded by $N_S = \lceil 3\sqrt{2\pi}(D+1)^{3/2}(\gamma\sqrt{1-\gamma^2/4})^{-D}\rceil$. By letting $\gamma \to 1$, then $N_S \to \lceil 3\sqrt{2\pi}(D+1)^{3/2}(4/3)^{D/2}\rceil$. Experimentally, Nguyen and Vidick [160] observed that the

Algorithm 1: The Nguyen-Vidick sieve

```
   Input: Basis $\mathbf{B}$ for a $D$-dimensional lattice and parameter $\gamma \in (0,1)$
   Output: Shortest vector $\mathbf{v}^*$ of the lattice.
1  Sample $L \subset \mathbb{R}^D$
2  while $L \neq \emptyset$ do
3      $L_0 \leftarrow L$, $L' \leftarrow \emptyset$, $S \leftarrow \{\mathbf{0}\}$, $R \leftarrow \max_{\mathbf{v}\in L} \|\mathbf{v}\|$
4      for each $\mathbf{v} \in L$ do
5          $\mathbf{w} \leftarrow$ GroverSearch($\mathbf{w} \in S : \|\mathbf{v} - \mathbf{w}\| \leq \gamma R$)
6          if $\mathbf{w} \neq$ NULL then                      // $\exists \mathbf{w} \in S : \|\mathbf{v} - \mathbf{w}\| \leq \gamma R$
7              $L' \leftarrow L' \cup \{\mathbf{v} - \mathbf{w}\}$
8          else                                        // $\nexists \mathbf{w} \in S : \|\mathbf{v} - \mathbf{w}\| \leq \gamma R$
9              $S \leftarrow S \cup \{\mathbf{v}\}$
10     $L \leftarrow L'$
11 return shortest vector $\mathbf{v}^*$ in $L_0$
```

size of S is upper-bounded by $\ln|S| \leq aD + b\ln D + c$ where $a = 0.163(\pm 0.017)$, $b = 0.102(\pm 0.65)$, and $c = 1.73(\pm 1.72)$ if $\gamma = 0.97$. In other words, $|S| \leq 2^{0.2352D + 0.102\log_2 D + 2.45}$.

2. In practice, one samples an initial list L of considerable size and runs the `NVSieve`. If the shortest vector is not found and the `NVSieve` thus fails, the whole procedure is restarted but with a larger initial list. Given numerical experiments from [160] and also conducted by us, an initial list L of size D times that of $|S| \leq 2^{0.2352D+0.102\log_2 D+2.45}$ suffices. Alternatively, $|L| = \lceil 3\sqrt{2\pi}(D+1)^{3/2}(4/3)^{D/2}\rceil$ also works.

3. As pointed out by Nguyen and Vidick [160], the list size $|L|$ decreases roughly by $(\gamma\sqrt{1-\gamma^2/4})^{-D}$ at each sieving step, provided the vectors in L are well distributed in $B_D(\gamma, R)$. Indeed, in Lines 6 to 9 from Algorithm 1, each $\mathbf{v} \in L$ is either selected to L' (reduced by $\mathbf{w}$) or to S, and thus $|L'| \approx |L| - |S|$ if there are few collisions ($\mathbf{v} - \mathbf{w} = \mathbf{0}$). Numerical experiments from [160, Figure 2] show that the number of collisions is negligible until $R/\lambda(\mathbf{B}) \approx 4/3$. Therefore, for most sieving steps, the size of L is reduced by the size of S, which is at most $2^{0.2352D+0.102\log_2 D+2.45}$.

4. As $\gamma \to 1$, the expected size of S decreases, while the number of sieving steps clearly increases. Nguyen and Vidick [160] used a contraction parameter $\gamma = 0.97$ in their simulations. By keeping $\gamma \approx 0.97$, we expect the upper-bound $|S| \leq 2^{0.2352D+0.102\log_2 D+2.45}$ to hold. Moreover, if γ is not too close to 1, we can abstract away the number of sieving steps by assuming that $|L|$ roughly decreases by $|S|$.

5. While the size of S fluctuates within a sieving step in Algorithm 1, there are other implementations of the `NVSieve` [127, 128] in which the list of centers S is sampled from L beforehand in every sieving step and, therefore, $|S|$ is kept constant.

7.1.2 Quantum oracle for Grover search

The `NVSieve` employs one search subroutine per sieve step, per list vector, which can be done using Grover's algorithm (Line 5 in Algorithm 1). For fixed $\mathbf{v} \in L$, the search is done over the centers S in order to find an element $\mathbf{w}$ such that $\|\mathbf{v} - \mathbf{w}\| \leq \gamma R$, where $R = \max_{\mathbf{v}\in L}\|\mathbf{v}\|$. Define the Boolean function $f_{\rm NV} : [|S|] \to \{0,1\}$ such that $f_{\rm NV}(i) = 1$ if and only if $\|\mathbf{v} - \mathbf{w}_i\| \leq \gamma R$. In order to use Grover search, we must implement the phase oracle $\mathcal{O}_{\rm NV} : |i\rangle \mapsto (-1)^{f_{\rm NV}(i)}|i\rangle$, as explained next.

Given any index $|i\rangle$ where $i \in [|S|]$, we start with one QRAM_S call to load $\mathbf{w}_i$ onto a (κD)-qubit ancillary register. The list of centers S is already loaded onto the QRAM at the beginning of every

Table 2: Amount of subroutines required to implement a phase oracle in each Grover search per sieve step in the following sieving algorithms: `NVSieve`, `NVSieve` with LSH/LSF, `GaussSieve`, and `GaussSieve` with LSH/LSF. The `NVSieve` requires only one type of Grover search, while the `GaussSieve` requires two types. All quantum adders, comparators, and multipliers are κ-bit out-of-place operations. Operations marked by $(*)$ have hybrid classical-quantum inputs and are thus cheaper. All subroutines include their inverse.

Sieve/Operations	QRAM	Adders	Multipliers	Extra CNOTs
`NVSieve`	1	$2D$	D	0
`NVSieve` + LSH/LSF	1	$2D$	D	0
`GaussSieve`	1	$4D-2$	$2D$	$2D\kappa+4$
	1	$D+1$	D^*	4
`GaussSieve` + LSH/LSF	1	$4D-2$	$2D$	$2D\kappa+4$
	1	$D+1$	D^*	4

sieve step and the resources required for one call are given in Lemma 9. Next we must compute the value of f_{NV}. Rewrite the inequality defining the Boolean function f_{NV} as

$$\sum_{j=1}^{D}(\mathbf{w}_i)_j(\mathbf{w}_i-2\mathbf{v})_j = \mathbf{w}_i\cdot(\mathbf{w}_i-2\mathbf{v}) \leq \gamma^2R^2 - \|\mathbf{v}\|^2.$$

In order to compute $\sum_{j=1}^{D}(\mathbf{w}_i)_j(\mathbf{w}_i-2\mathbf{v})_j$, we first compute $\mathbf{w}_i-2\mathbf{v}$ using D parallel κ-bit out-of-place adders with the classical input $2\mathbf{v}$. At this point the quantum registers hold $|i\rangle|\mathbf{w}_i\rangle|\mathbf{w}_i-2\mathbf{v}\rangle$. Next, all the terms $(\mathbf{w}_i)_j(\mathbf{w}_i-2\mathbf{v})_j$, $j\in[D]$, are computed using D parallel κ-bit out-of-place multipliers. This yields the quantum registers $|i\rangle|\mathbf{w}_i\rangle|\mathbf{w}_i-2\mathbf{v}\rangle\bigotimes_{j=1}^{D}|(\mathbf{w}_i)_j(\mathbf{w}_i-2\mathbf{v})_j\rangle$. For the next step, all D terms $(\mathbf{w}_i)_j(\mathbf{w}_i-2\mathbf{v})_j$ are summed in depth $\lceil\log_2 D\rceil$ by using $D-1$ κ-bit out-of-place adders. Finally, we employ a κ-bit comparator (which counts as an κ-bit adder) between the quantum register holding $\sum_{j=1}^{D}(\mathbf{w}_i)_j(\mathbf{w}_i-2\mathbf{v})_j$ and the classical input $\lambda := \gamma^2R^2-\|\mathbf{v}\|^2$, but the output register of the comparator is initialised in the $|-\rangle$ state instead of the $|0\rangle$ state. This procedure introduces the phase $(-1)^{f_{\text{NV}}(i)}$ as wanted. We summarise the whole chain of operations as follows:

$$\begin{aligned}
&|-\rangle|i\rangle|0\rangle^{\otimes\kappa(2+3D)}\\
\xrightarrow{\mathsf{QRAM}_S}\ &|-\rangle|i\rangle|\mathbf{w}_i\rangle|0\rangle^{\otimes\kappa(2+2D)}\\
\xrightarrow{D\text{ adders}}\ &|-\rangle|i\rangle|\mathbf{w}_i\rangle|\mathbf{w}_i-2\mathbf{v}\rangle|0\rangle^{\otimes\kappa(2+D)}\\
\xrightarrow{D\text{ multipliers}}\ &|-\rangle|i\rangle|\mathbf{w}_i\rangle|\mathbf{w}_i-2\mathbf{v}\rangle\left(\bigotimes_{j=1}^{D}|(\mathbf{w}_i)_j(\mathbf{w}_i-2\mathbf{v})_j\rangle\right)|0\rangle^{\otimes2\kappa}\\
\xrightarrow{D-1\text{ adders}}\ &|-\rangle|i\rangle|\mathbf{w}_i\rangle|\mathbf{w}_i-2\mathbf{v}\rangle\left(\bigotimes_{j=1}^{D}|(\mathbf{w}_i)_j(\mathbf{w}_i-2\mathbf{v})_j\rangle\right)|\mathbf{w}_i^\top(\mathbf{w}_i-2\mathbf{v})\rangle|0\rangle^{\otimes\kappa}\\
\xrightarrow{1\text{ adder}}\ &(-1)^{f_{\text{NV}}(i)}|-\rangle|i\rangle|\mathbf{w}_i\rangle|\mathbf{w}_i-2\mathbf{v}\rangle\left(\bigotimes_{j=1}^{D}|(\mathbf{w}_i)_j(\mathbf{w}_i-2\mathbf{v})_j\rangle\right)|\mathbf{w}_i^\top(\mathbf{w}_i-2\mathbf{v})\rangle|\mathbf{w}_i^\top(\mathbf{w}_i-2\mathbf{v})-\lambda\rangle.
\end{aligned}$$

At the end of this chain of operations, after the phase $(-1)^{f_{\text{NV}}(i)}$ has been applied, we uncompute all the $2D$ adders, D multipliers, and one QRAM_S call using their suitable inverses. Even though all the extra ancillae required for adders, multipliers, and QRAM are not made explicit in the arguments above, dirty ancillae are kept throughout the computations in order to ease their inverses. In Table 2, we summarise all the arithmetic and memory modules required in the phase oracle $\mathcal{O}_{\text{NV}}$.

Algorithm 2: The Nguyen-Vidick sieve with LSH

```
Input: Basis $\mathbf{B}$ for a $D$-dimensional lattice, parameters $\gamma \in (0,1)$, $k$, and $t$, hash family $\mathcal{H}$.
Output: Shortest vector $\mathbf{v}^*$ of the lattice.
1  Sample $L \subset \mathbb{R}^D$
2  while $L \neq \emptyset$ do
3      $L_0 \leftarrow L$, $L' \leftarrow \emptyset$, $S \leftarrow \{\mathbf{0}\}$, $R \leftarrow \max_{\mathbf{v}\in L} \|\mathbf{v}\|$
4      Initialise $t$ empty hash tables $\mathcal{T}_1, \dots, \mathcal{T}_t$ and sample $k \cdot t$ random hash functions $h_{i,j} \in \mathcal{H}$
5      For each $i \in [t]$, add $\mathbf{0}$ to the bucket $\mathcal{T}_i[h_i(\mathbf{0})]$ in the hash table $\mathcal{T}_i$
6      for each $\mathbf{v} \in L$ do
7          $C \leftarrow \bigcup_{i=1}^{t} \mathcal{T}_i[h_i(\mathbf{v})]$ is the list of candidate vectors
8          Construct a QRAM for $C$
9          $\mathbf{w} \leftarrow$ GroverSearch($\mathbf{w} \in C : \|\mathbf{v} - \mathbf{w}\| \leq \gamma R$)
10         if $\mathbf{w} \neq$ NULL then                // $\exists \mathbf{w} \in C : \|\mathbf{v} - \mathbf{w}\| \leq \gamma R$
11             $L' \leftarrow L' \cup \{\mathbf{v} - \mathbf{w}\}$
12         else                                  // $\nexists \mathbf{w} \in C : \|\mathbf{v} - \mathbf{w}\| \leq \gamma R$
13             $S \leftarrow S \cup \{\mathbf{v}\}$
14             For each $i \in [t]$, add $\mathbf{v}$ to the bucket $\mathcal{T}_i[h_i(\mathbf{v})]$ in the hash table $\mathcal{T}_i$
15     $L \leftarrow L'$
16 return shortest vector $\mathbf{v}^*$ in $L_0$
```

7.1.3 Using LSH/LSF in the Nguyen-Vidick sieve

Locality-sensitive hashing and filtering can be used to speed up sieving, particularly the `NVSieve` [160]. The main idea of employing nearest-neighbour-search techniques in the `NVSieve` is to preprocess the list of centers S and thus replace the brute-force list search over $\mathbf{w} \in S$ with a smaller list of probable reducible candidates. As described in Section 2.2, we sample $k \cdot t$ random hash function $h_{i,j} \in \mathcal{H}$ from a suitable hash family $\mathcal{H}$, and using the AND and OR-compositions, introduce t hash tables, each with an exponential number of buckets in the parameter k (2^k buckets in angular LSH and $2^{k\sqrt{D}}$ buckets in spherical LSH). Each vector $\mathbf{w} \in S$ is then added to its corresponding bucket $\mathcal{T}_i[h_i(\mathbf{v})]$ labelled by the hash $h_i(\mathbf{v})$ for each hash table $\mathcal{T}_i$. Afterwards, given $\mathbf{v} \in L$, only vectors in buckets $\mathcal{T}_1[h_1(\mathbf{v})], \dots, \mathcal{T}_t[h_t(\mathbf{v})]$ are considered as possible candidates for reduction. The search space is thus considerable reduced and many of far-away vectors in S to $\mathbf{v}$ are ignored. The `NVSieve` with LSH is described in Algorithm 2. A similar procedure applies to LSF: $k \cdot t$ random filter functions $f_{i,j} \in \mathcal{F}$ from a suitable filter family $\mathcal{F}$ are sampled and employed to add vectors onto t different filtered buckets $\mathcal{B}_1, \dots, \mathcal{B}_t$. A vector $\mathbf{w} \in S$ is added onto the bucket $\mathcal{B}_i$ if and only if it passes through all filters $f_{i,1}, \dots, f_{i,k}$. Afterwards, a query $\mathbf{v} \in L$ is answered by recovering all the filters that $\mathbf{v}$ passes through. We note that insertions into buckets and searching over relevant filters might use filters with different internal parameters (an example being the parameter α is spherical LSF). The different between LSH and LSF is that, in the second case, each hash table is reduced to a single bucket of vectors that survived the filters. Another difference is the use of random product codes to efficiently find all buckets that contain a given vector.

Apart from searching over the buckets $\mathcal{T}_1[h_1(\mathbf{v})], \dots, \mathcal{T}_t[h_t(\mathbf{v})]$ for a reducible vector using Grover's algorithm, another big difference between the classical and quantum versions of the `NVSieve` with LSH is obtaining the list of candidates vectors $C = \bigcup_{i=1}^{t} \mathcal{T}_i[h_i(\mathbf{v})]$ in the first place. Whereas classically the search over C can be done while sequentially visiting all the t buckets, to retain the quadratic quantum advantage, we must first classically gather all the indices (or hashes) of the vectors in the buckets $\mathcal{T}_1[h_1(\mathbf{v})], \dots, \mathcal{T}_t[h_t(\mathbf{v})]$, after which we can perform Grover's search over these vectors. If $C = \{\mathbf{w}_{j_1}, \mathbf{w}_{j_2}, \dots, \mathbf{w}_{j_{|C|}}\}$, then we start with the state $|C|^{-1/2} \sum_{i=1}^{|C|} |i\rangle$ within Grover's search. To proceed, we would like to use QRAM to map $|i\rangle|0\rangle^{\otimes \kappa D} \mapsto |i\rangle|\mathbf{w}_{j_i}\rangle$. This can be done using the QRAM of Figure 4 by classically ordering the hashes $\{j_1, j_2, \ldots, j_{|C|}\}$ so that a classically controlled CNOT is applied based on the content $\mathbf{w}_{j_i}$, which is accessed via a RAM call. The phase oracle $\mathcal{O}_{\mathrm{NV}} : |i\rangle \mapsto (-1)^{f_{\mathrm{NV}}(i)}|i\rangle$, where $f_{\mathrm{NV}} : [|C|] \to \{0,1\}$ is defined by $f_{\mathrm{NV}}(i) = 1$ if and only if $\|\mathbf{v} - \mathbf{w}_{j_i}\| \leq \gamma R$, is hence implemented initially as

$$|i\rangle|0\rangle^{\otimes \kappa D} \xrightarrow{\mathsf{QRAM}_C} |i\rangle|\mathbf{w}_{j_i}\rangle, \tag{7}$$

after which the remaining addition and multiplication operations explained in the previous section are performed (plus overall uncomputation). The phase oracle $\mathcal{O}_{\mathrm{NV}}$ thus requires one QRAM call one of size $|C|$ and $|C|$ RAM calls of size $|L|$. All the required subroutines to implement the phase oracle are summarised in Table 2.

The need to take QRAM into consideration means that the use of Grover's search in the `NVSieve` does not improve its asymptotic scaling, since gathering the list of candidates C for each $\mathbf{v} \in L$ takes $O(|L| \cdot |C|)$ time. The only improvement is moving the more expensive norm computation into the Grover's search, so that the classical cost of $O(D \cdot |L| \cdot |C|)$ per sieving step becomes a classical-quantum cost of $O(|L| \cdot |C| + D \cdot |L| \cdot \sqrt{|C|})$.

7.1.4 NVSieve with angular LSH

When employing angular LSH, we hash each vector into a k-bit string for each of the t hash tables. Each hash table has thus 2^k buckets. We choose $k = \log_{3/2} t - \log_{3/2} \ln(1/\varepsilon)$ so that nearby vectors collide with high probability. Hashing one vector requires computing $\mathbf{a}_i \cdot \mathbf{v}$ for all $i \in [k]$, so kD multiplications and $k(D-1)$ additions. As pointed out by Laarhoven [127], it is possible to employ sparse vectors $\mathbf{a}_i$ for the angular hash functions while still maintaining the same performance [2, 3, 135]. Laarhoven [127] employed vectors $\mathbf{a}_i$ with just two non-zero entries, therefore we require $2k$ multiplications and k additions to hash $\mathbf{v}$. The time spent hashing all vectors in the list L into the t hash tables is thus $O(k \cdot |L| \cdot t)$, or more precisely, it requires $2k \cdot |L| \cdot t$ multiplications and $k \cdot |L| \cdot t$ additions. On the other hand, the list of candidates on Line 7 has size $|C| \approx |S| \cdot p_2^*$, where p_2^* is the average probability that a non-reducing vector collides with another vector in at least one of the t hash tables (Equation (1)).

Classical complexity. The classical time spent searching over C is $O(D \cdot |S| \cdot |L| \cdot p_2^*)$ per sieving step. More precisely, according to Table 2 (the number of classical arithmetic operations is the same as in the quantum case), searching over C requires $D \cdot |L| \cdot |C| = D \cdot |L| \cdot |S| \cdot p_2^*$ multiplications and $2D \cdot |L| \cdot |C| = 2D \cdot |L| \cdot |S| \cdot p_2^*$ additions per sieving step. The number of hash tables t is determined by balancing the time hashing $O(k \cdot |L| \cdot t)$ with the time searching $O(D \cdot |L| \cdot |S| \cdot p_2^*)$. Asymptotically over all sieving steps, t is determined by taking $|L| = |S| = (4/3)^{D/2+o(D)}$ and approximating $p_2^* \approx t \cdot 2^{-\beta D + o(D)}$, where β is given by Equation (2). We must then have that $(4/3)^{D/2+o(D)} \cdot 2^{-\beta D+o(D)} = 2^{o(D)} \implies \beta = \frac{1}{2}\log_2(4/3)$, which yields $t \approx 2^{0.129043D}$. Therefore, by using $t \approx 2^{0.129043D}$ hash tables and a hash length of $k \approx 0.220600D$, the overall time and space complexities are $\widetilde{O}(|L| \cdot t) = 2^{0.336562D+o(D)}$ [127, 128].

Quantum complexity. The quantum time searching over C is $O(D \cdot |L|\sqrt{|S| \cdot p_2^*})$ per sieving step. The number of hash tables t is determined by balancing the time hashing $O(k \cdot |L| \cdot t)$ with the time searching $O(D \cdot |L|\sqrt{|S| \cdot p_2^*})$. Asymptotically, t is determined by taking $|L| = |S| = (4/3)^{D/2+o(D)}$ and approximating $p_2^* \approx t \cdot 2^{-\beta D+o(D)}$, so that $(4/3)^{D/4+o(D)} \cdot 2^{-\beta D/2+o(D)} = \sqrt{t} \implies \beta = \frac{1}{2}\log_2(4/3) - \frac{1}{D}\log_2 t$. This yields $t \approx 2^{0.078430D}$. Therefore, by using $t \approx 2^{0.078430D}$ hash tables and a hash length of $k \approx 0.134077D$, the overall time and space complexities are $\widetilde{O}(|L| \cdot t) = 2^{0.285949D+o(D)}$ [130, 128].

7.1.5 NVSieve with spherical LSH

The complexity analysis of the `NVSieve` with spherical LSH (also known as `SphereSieve` [130]) is similar to `NVSieve` with angular LSH. When employing spherical LSH, we hash each vector into a string in $[2^{\sqrt{D}}]^k$ for each of the t hash tables. Each hash table has thus $2^{k\sqrt{D}}$ buckets. We choose $k = 6(\ln t - \ln\ln(1/\varepsilon))/\sqrt{D}$ so that nearby vectors collide with high probability. Hashing one vector requires comparing $\langle \mathbf{v}, \mathbf{g}_{ij}\rangle \geq D^{1/4}$ for $i \in [2^{\sqrt{D}}]$ and $j \in [k]$. The time spent hashing all vectors in the list L into the t hash tables is thus $O(D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot |L|)$, or more precisely, it requires $D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot |L|$ multiplications and $D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot |L|$ additions. On the other hand, the list of candidates on Line 7 has size $|C| \approx |S| \cdot p_2^*$, where p_2^* is the average probability that a non-reducing vector collides with another vector in at least one of the t hash tables (Equation (3)).

Classical complexity. Classically searching over C requires $D \cdot |L| \cdot |S| \cdot p_2^*$ multiplications and $D \cdot |L| \cdot |S| \cdot p_2^*$ additions per sieving step. The number of hash tables is determined by balancing the time hashing $O(D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot |L|)$ and the time searching $O(D \cdot |L| \cdot |S| \cdot p_2^*)$. Asymptotically, t is determined by taking $|L| = |S| = (4/3)^{D/2+o(D)}$ and approximating $p_2^* \approx 2^{-\beta D+o(D)}$, where β is given by Equation (4). Hence $(4/3)^{D/2+o(D)} \cdot 2^{-\beta D+o(D)} = t \implies \beta = \frac{1}{2}\log_2(4/3) - \frac{1}{D}\log_2 t$, which yields $t \approx 2^{0.089624D}$. Therefore, by using $t \approx 2^{0.089624D}$ hash tables and $k \approx 0.372737\sqrt{D}$, the time and space complexities are $2^{0.297143D+o(D)}$ [129, 128].

Quantum complexity. Quantumly searching over C requires $O(D \cdot |L|\sqrt{|S| \cdot p_2^*})$ time. The number of hash tables is determined by balancing the time hashing $O(D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot |L|)$ and the time searching $O(D \cdot |L|\sqrt{|S| \cdot p_2^*})$. Asymptotically, t is determined by taking $|L| = |S| = (4/3)^{D/2+o(D)}$ and approximating $p_2^* \approx 2^{-\beta D+o(D)}$, so that $(4/3)^{D/4+o(D)} \cdot 2^{-\beta D/2+o(D)} = t \implies \beta = \frac{1}{2}\log_2(4/3) - \frac{2}{D}\log_2 t$. This yields $t \approx 2^{0.059581D}$. Therefore, by using $t \approx 2^{0.059581D}$ hash tables and $k \approx 0.247792\sqrt{D}$, the overall time and space complexities are $2^{0.267100D+o(D)}$ [130, 128].

7.1.6 NVSieve with spherical LSF

The complexity analysis of the `NVSieve` with spherical LSF is a bit different than with LSH, the main reason being that each filter bucket covers an equally large region of $\mathbb{R}^D$, which simplifies the analysis. As shown in [32], fixing $k = 1$ concatenated filters per bucket is usually optimal, as it allows a larger α parameter (see Equation (5)). On the other hand, the number of filter buckets t is chosen so that nearby vectors collide with high probability $p_1^* \geq 1 - \varepsilon$, where p_1^* is given by Equation (6), meaning that $t = \ln(1/\varepsilon)/\mathcal{W}_D(\alpha, \alpha, \pi/3)$. Each vector is on average contained in $t \cdot \mathcal{C}_D(\alpha)$ buckets, meaning there are $|S| \cdot t \cdot \mathcal{C}_D(\alpha)$ vectors in total in all buckets and $|S| \cdot \mathcal{C}_D(\alpha)$ vectors in each bucket on average. The list of candidates on Line 7 has size $|C| \approx |S| \cdot t \cdot \mathcal{C}_D(\alpha)^2$. Inserting vectors into relevant filters requires an efficient oracle (as mentioned in Section 2.2.3). Becker et al. [32] proposed such an efficient oracle, called `EfficientListDecoding`, using random product codes. According to [32, Lemma 5.1] (see Fact 5), the time complexity of such an oracle is mainly due to visiting at most $2\log_2 D \cdot t \cdot \mathcal{C}_D(\alpha)$ nodes for a pruned enumeration, which requires mostly addition-like operations. We thus approximate the time to insert all the vectors in L into relevant filters by $2\log_2 D \cdot |L| \cdot t \cdot \mathcal{C}_D(\alpha)$ additions.

Classical complexity. The classical time spent searching over C requires $D \cdot |L| \cdot |S| \cdot t \cdot \mathcal{C}_D(\alpha)^2$ additions and multiplications per sieving step. Moreover, we also need to retrieve the relevant filters of $\mathbf{v}$ before performing the search over C. Retrieving the relevant filters of all vectors in L requires $2\log_2 D \cdot |L| \cdot t \cdot \mathcal{C}_D(\alpha)$ additions per sieving step. The parameter α is chosen in order to minimise the time complexity coming from filtering and from searching, $O(\log D \cdot |L| \cdot t \cdot \mathcal{C}_D(\alpha) + D \cdot |L| \cdot |S| \cdot t \cdot \mathcal{C}_D(\alpha)^2)$. Asymptotically, we approximate $t = O(1/\mathcal{W}_D(\alpha, \alpha, \pi/3))$ (see Section 2.2.3). On the other hand, $\mathcal{C}_D(\alpha) = \mathrm{poly}(D)(1-\alpha^2)^{D/2}$ [156, Lemma 4.1] and $\mathcal{W}_D(\alpha, \alpha, \theta) = \mathrm{poly}(D)(1 - 2\alpha^2/(1+\cos\theta))^{D/2}$ [32, Lemma 2.2]. Together with $|L| = (4/3)^{D/2+o(D)}$, this means that the total time complexity over all sieving steps is

$$\widetilde{O}\left(\frac{|L| \cdot \mathcal{C}_D(\alpha)(1 + |L| \cdot \mathcal{C}_D(\alpha))}{\mathcal{W}_D(\alpha, \alpha, \pi/3)}\right) = \widetilde{O}\left(\left(\frac{4(1-\alpha^2)}{3-4\alpha^2}\right)^{D/2}\left(1 + \left(\frac{4(1-\alpha^2)}{3}\right)^{D/2}\right)\right).$$

The high-order term is minimised by taking $\alpha = 1/2$. Therefore, the time complexity is $(3/2)^{D/2+o(D)} \approx 2^{0.292481D+o(D)}$ by choosing $\alpha = 1/2$, $k = 1$, and $t = (3/2)^{D/2+o(D)}$ [32, 128]. The space complexity is also $(3/2)^{D/2+o(D)}$. The list of candidates, i.e., the list of vectors that collide with a given vector, has average size $|C| = |L| \cdot t \cdot \mathcal{C}_D(\alpha)^2 = (9/8)^{D/2+o(D)}$.

Quantum complexity. The quantum time spent comparing a given vector to other vectors colliding in one of the filters is now $O(D\sqrt{|S| \cdot t \cdot \mathcal{C}_D(\alpha)^2})$. The total time complexity of one sieving step with list L is thus $O(\log D \cdot |L| \cdot t \cdot \mathcal{C}_D(\alpha) + D \cdot |L| \cdot |S|^{1/2} \cdot t^{1/2} \cdot \mathcal{C}_D(\alpha))$. The α parameter is chosen in order to minimise the classical hashing time plus the quantum searching time. Asymptotically, the approximations $t = O(1/\mathcal{W}_D(\alpha, \alpha, \pi/3))$, $\mathcal{C}_D(\alpha) = \mathrm{poly}(D)(1-\alpha^2)^{D/2}$, and $\mathcal{W}_D(\alpha, \alpha, \theta) = \mathrm{poly}(D)(1-2\alpha^2/(1+\cos\theta))^{D/2}$, together with $|L| = (4/3)^{D/2+o(D)}$, yield the total time complexity over all sieving steps of

$$\widetilde{O}\left(\frac{|L| \cdot \mathcal{C}_D(\alpha)}{\mathcal{W}_D(\alpha,\alpha,\pi/3)} + \frac{|L|^{3/2} \cdot \mathcal{C}_D(\alpha)}{\sqrt{\mathcal{W}_D(\alpha,\alpha,\pi/3)}}\right) = \widetilde{O}\left(\left(\frac{4(1-\alpha^2)}{3-4\alpha^2}\right)^{D/2} + \left(\frac{8(1-\alpha^2)}{3\sqrt{3-4\alpha^2}}\right)^{D/2}\right).$$

The high-order term is minimised by taking $\alpha = \sqrt{3}/4$. Hence the time complexity is $(13/9)^{D/2+o(D)} \approx 2^{0.265257D+o(D)}$ by choosing $\alpha = \sqrt{3}/4$, $k = 1$, and $t = (4/3)^{D/2+o(D)}$ [128]. The space complexity is also $(13/9)^{D/2+o(D)}$. The list of candidate vectors has average size $|C| = |L| \cdot t \cdot \mathcal{C}_D(\alpha)^2 = (13/12)^{D+o(D)}$.

7.2 The GaussSieve

A few years after the work of Nguyen and Vidick, Micciancio and Voulgaris [156] presented `ListSieve`, a probabilistic algorithm that provably finds the shortest vector with a high probability in $2^{3.199D+o(D)}$ time and $2^{1.325D+o(D)}$ space, and a heuristic variant called `GaussSieve`, which we now focus on and is described in Algorithm 3. The `GaussSieve` starts with an empty list L and keeps adding shorter lattice vectors to it. At each sieve step, a new lattice vector $\mathbf{v}$ is reduced with all the points in the list L. By this we mean the rule:

$$\text{Reduce } \mathbf{v} \text{ with } \mathbf{w}: \quad \text{if } \|\mathbf{v} \pm \mathbf{w}\| < \|\mathbf{v}\| \text{ then } \mathbf{v} \leftarrow \mathbf{v} \pm \mathbf{w}.$$

The difference between both sieves is that, in the `ListSieve`, the reduced vector is then added to the list, meaning that vectors in L never change, while in the `GaussSieve`, we also attempt to reduce the vectors in L with $\mathbf{v}$ before adding $\mathbf{v}$ to L. In other words, in the `GaussSieve`, for all vectors $\mathbf{v}, \mathbf{w} \in L$ such that $\min(\|\mathbf{v} \pm \mathbf{w}\|) < \max(\|\mathbf{v}\|, \|\mathbf{w}\|)$, the longer of $\mathbf{v}$ and $\mathbf{w}$ is replaced with the shorter of $\mathbf{v} \pm \mathbf{w}$. Consequently, all pairs of vectors in the list are always pairwise reduced: $\forall \mathbf{v}, \mathbf{w} \in L : \min(\|\mathbf{v} \pm \mathbf{w}\|) \geq \max(\|\mathbf{v}\|, \|\mathbf{w}\|)$. Thus any pair of vectors in the list always form a Gauss reduced basis for a two dimensional lattice, and thus the angle between any two list points is at least $\pi/3$ and the list forms a good spherical code. It follows that the size of the list never exceeds the kissing constant τ_D in D dimensions. Therefore the list size (and thus the space complexity of the `GaussSieve`) is bounded by $2^{0.401D}$ in theory and $2^{0.208D}$ in practice, corresponding to the asymptotic upper and lower bounds on τ_D [111]. In contrast, there are no known bounds on the time complexity of the `GaussSieve`, since the list L can grow or shrink at any time. One might guess that the time complexity is quadratic in the list size since at each sieving step every pair of vectors $\mathbf{v}, \mathbf{w} \in L$ is compared at least once to check for possible reductions. Furthermore, the asymptotic behaviour of the `GaussSieve` is similar to that of the `NVSieve` [156]. A natural conjecture is that the `GaussSieve` has time complexity $\widetilde{O}(|L|^2) = 2^{0.415D+o(D)}$.

7.2.1 Numerical experiments and heuristic assumptions

Once again, the asymptotic complexity hides a lot of operations when doing a resource estimate. The overall analysis of `GaussSieve` is more complicate than the `NVSieve`, since the size of the list L can both increase and decrease, which hinders a bound on time complexity. Moreover, the search loops in Lines 7 and 9 are performed in an exhaustive manner, meaning that a search will be attempted

Algorithm 3: The `GaussSieve`

```
Input: Basis $\mathbf{B}$ for a $D$-dimensional lattice and termination constant $\zeta$.
Output: Shortest vector $\mathbf{v}^*$ of the lattice.
1  $L \leftarrow \emptyset$, $S \leftarrow \emptyset$, $K \leftarrow 0$
2  while $K < \zeta$ do
3      if $S = \emptyset$ then
4          $\mathbf{v} \leftarrow$ SampleKlein($\mathbf{B}$)
5      else
6          $\mathbf{v} \leftarrow S$.pop()
7      while $\mathbf{w} \leftarrow$ GroverSearch($\mathbf{w} \in L : \|\mathbf{v} \pm \mathbf{w}\| < \|\mathbf{v}\|$) and $\mathbf{w} \neq$ NULL do
8          Reduce $\mathbf{v}$ with $\mathbf{w}$
9      while $\mathbf{w} \leftarrow$ GroverSearch($\mathbf{w} \in L : \|\mathbf{w} \pm \mathbf{v}\| < \|\mathbf{w}\|$) and $\mathbf{w} \neq$ NULL do
10         $L \leftarrow L \setminus \{\mathbf{w}\}$
11         Reduce $\mathbf{w}$ with $\mathbf{v}$
12         if $\mathbf{w} \neq \mathbf{0}$ then
13             $S \leftarrow S \cup \{\mathbf{w}\}$
14     if $\mathbf{v}$ has changed and $\mathbf{v} \neq \mathbf{0}$ then
15         $S$.push($\mathbf{v}$)
16     if $\mathbf{v} = \mathbf{0}$ then
17         $K \leftarrow K + 1$
18     else
19         $L \leftarrow L \cup \{\mathbf{v}\}$
20 return shortest vector $\mathbf{v}^*$ in $L$
```

while there are solutions. Nonetheless, it is still possible to gather average trends and bounds through heuristic assumptions and numerical experiments. In the following, we list several observations that shall be useful in forming assumptions.

1. Schneider [181] noticed that `GaussSieve`'s performance in terms of runtime, iterations, list size, and collisions was not affected by the type of the underlying lattice (ideal, cyclic, and random).

2. Micciancio and Voulgaris [156] proved that the list size $|L|$ never exceeds the kissing number τ_D, which is defined as the highest number of points that can be placed on a D-dimensional sphere such that the angle between any two points is at least $\pi/3$. This theoretically bounds $|L|$ by $\tau_D \leq 2^{0.401D+o(D)}$. However, Micciancio and Voulgaris [156] numerically observed that the maximum list size grows approximately as $2^{0.2D}$, which matches lower bounds on the kissing number $\tau_D \geq 2^{0.2075D+o(D)}$ [61]. A plausible assumption is the maximum list size to be bounded by a lower bound on the kissing number, e.g., $\tau_D \geq (1+o(1))\frac{\sqrt{3\pi}}{4\sqrt{2}}\ln(3/2)D^{3/2}(4/3)^{D/2}$ [75]. From a more numerical perspective, Schneider [181] reported a maximum list size of $2^{0.2D+2.8}$, while Mariano et al. [149] reported a maximum list size of $2^{0.199D+2.149}$. We independently report a maximum list size of $2^{0.193D+2.325}$.

3. Schneider [181] observed that the number of times a newly sampled vector from Klein's algorithm was reduced by the list vectors and the number of vectors removed from the list L and pushed to the stack S were approximately 10 times the maximum list size. This means that on average the first search loop (Line 7) is performed 10 times. This observation was independently confirmed by us. The number of solutions to Grover's search in Line 7 varies greatly. A more pessimistic assumption is to take $M = 1$ or $M = 2$ solutions for each of the first 9 calls, while the 10-th call

has $M = 0$ solutions. On the other hand, the second search loop (Line 9) is performed only once with $M = 0$ number of solutions on the vast majority of cases.

4. The number of sieving steps is roughly 10 times the maximum list size (see [181, Figures 1 and 2]). Mariano et al. [149] numerically reported the number of iterations I to grow as $2^{0.283D+0.335}$, while we obtained a growth of $2^{0.283D+0.491}$.

5. A natural termination criteria proposed by Micciancio and Voulgaris [156] is to stop after a certain number ζ of collisions. A conservative option for ζ adopted by [156] is to set it as 10% of the list size. The authors[1] also used an alternative criteria of $\zeta = 500$, which we independently checked to be enough to find the shortest vector. Under such criteria, the list size does not grow much beyond the point where a shortest vector is found.

6. The list size $|L|$ starts from 0 and quickly grows to an asymptote which, according to the previous point, roughly corresponds to the maximum list size. Meanwhile, collisions rarely occur before the shortest vector is found, after which the number of collisions quickly grows until the exit-condition is reached. The list size stays above 90% of the maximum list size (i.e., the list size at the moment a shortest vector is found) for more than 95% the number of iterations for large enough dimensions ($D > 70$).

7.2.2 Quantum oracle for Grover search

Similarly to the `NVSieve`, the two search steps in the `GaussSieve` (Lines 7 and 9 in Algorithm 3) can be performed using Grover's algorithm. Namely, given an ordered list $L = \{\mathbf{w}_1, \mathbf{w}_2, \dots\}$ and a fixed element $\mathbf{v} \in L$,

1. Find an index $i \in [|L|]$ such that $\|\mathbf{v} \pm \mathbf{w}_i\| < \|\mathbf{v}\| \iff \mathbf{w}_i \cdot (\mathbf{w}_i \pm 2\mathbf{v}) < 0$;
2. Find an index $i \in [|L|]$ such that $\|\mathbf{v} \pm \mathbf{w}_i\| < \|\mathbf{w}_i\| \iff |\mathbf{v} \cdot \mathbf{w}_i| \geq \|\mathbf{v}\|^2/2$.

Define the Boolean function $f_{\text{gauss}} : [|L|] \to \{0,1\}$ by $f_{\text{gauss}}(i) = 1$ if and only if either $\mathbf{w}_i \cdot (\mathbf{w}_i + 2\mathbf{v}) < 0$ or $\mathbf{w}_i \cdot (\mathbf{w}_i - 2\mathbf{v}) < 0$. Similarly, let $g_{\text{gauss}} : [|L|] \to \{0,1\}$ be such that $g_{\text{gauss}}(i) = 1$ if and only if $|\mathbf{v} \cdot \mathbf{w}_i| \geq \|\mathbf{v}\|^2/2$. In order to use Grover search in Line 7, we must construct the phase oracle $\mathcal{O}^{(1)}_{\text{gauss}} : |i\rangle \mapsto (-1)^{f_{\text{gauss}}(i)}|i\rangle$, while the Grover search in Line 9 requires the phase oracle $\mathcal{O}^{(2)}_{\text{gauss}} : |i\rangle \mapsto (-1)^{g_{\text{gauss}}(i)}|i\rangle$. We now describe how they can be constructed.

Phase oracle $\mathcal{O}^{(1)}_{\text{gauss}}$. The construction is similar to the one for the `NVSieve`. We assume that the list L is already stored in QRAM. Given any index $|i\rangle$ where $i \in [|L|]$, the first step is to load $\mathbf{w}_i$ onto a (κD)-qubit register using one QRAM_L call (Lemma 9). Since we must check for two conditions, $\mathbf{w}_i \cdot (\mathbf{w}_i + 2\mathbf{v}) < 0$ or $\mathbf{w}_i \cdot (\mathbf{w}_i - 2\mathbf{v}) < 0$, we copy $|\mathbf{w}_i\rangle$ onto another (κD)-qubit ancillary register using κD CNOTs. We then use $2D$ κ-bit out-of-place adders in parallel to get $|i\rangle|\mathbf{w}_i\rangle^{\otimes 2}|\mathbf{w}_i + 2\mathbf{v}\rangle|\mathbf{w}_i - 2\mathbf{v}\rangle$. Next, all the terms $(\mathbf{w}_i)_j(\mathbf{w}_i \pm 2\mathbf{v})_j$, $j \in [D]$, are computed in parallel using $2D$ κ-bit out-of-place multipliers. Then, all D terms $(\mathbf{w}_i)_j(\mathbf{w}_i + 2\mathbf{v})_j$ are summed in depth $\lceil \log_2 D \rceil$ by using $D - 1$ κ-bit out-of-place adders, and similarly for the terms $(\mathbf{w}_i)_j(\mathbf{w}_i - 2\mathbf{v})_j$. In order to check whether $\mathbf{w}_i \cdot (\mathbf{w}_i \pm 2\mathbf{v})$ is smaller than 0, it suffices to consider its highest-order bit. Since at most one of the conditions $\mathbf{w}_i \cdot (\mathbf{w}_i \pm 2\mathbf{v}) < 0$ can be true, we simply compute the parity of their high-bits instead of their logical OR. Thus, by applying two CNOTs controlled on the high-bits of $\mathbf{w}_i \cdot (\mathbf{w}_i \pm 2\mathbf{v})$ onto a qubit in the $|-\rangle$ state, we implement the phase $(-1)^{f_{\text{gauss}}(i)}$. After that, we uncompute all the arithmetic operations, copying of $|\mathbf{w}_i\rangle$, and QRAM call. The amount of submodules is summarised in Table 2.

[1]See Appendix B of the unpublished version.

Algorithm 4: The `GaussSieve` with LSH

```
Input: Basis $\mathbf{B}$ for a $D$-dimensional lattice, termination constant $\zeta$, parameters $k$ and $t$, hash
       family $\mathcal{H}$
Output: Shortest vector $\mathbf{v}^*$ of the lattice.
1  $L \leftarrow \emptyset$, $S \leftarrow \emptyset$, $K \leftarrow 0$
2  Initialise $t$ empty hash tables $\mathcal{T}_1, \dots, \mathcal{T}_t$ and sample $k \cdot t$ random hash functions $h_{i,j} \in \mathcal{H}$
3  while $K < \zeta$ do
4      if $S = \emptyset$ then
5          $\mathbf{v} \leftarrow$ SampleKlein($\mathbf{B}$)
6      else
7          $\mathbf{v} \leftarrow S$.pop()
8      $C \leftarrow \bigcup_{i=1}^{t} \mathcal{T}_i[h_i(\mathbf{v})]$ is the list of candidate vectors
9      Construct a QRAM for $C$
10     while $\mathbf{w} \leftarrow$ GroverSearch($\mathbf{w} \in C : \|\mathbf{v} \pm \mathbf{w}\| < \|\mathbf{v}\|$) and $\mathbf{w} \neq$ NULL do
11         Reduce $\mathbf{v}$ with $\mathbf{w}$
12     while $\mathbf{w} \leftarrow$ GroverSearch($\mathbf{w} \in C : \|\mathbf{w} \pm \mathbf{v}\| < \|\mathbf{w}\|$) and $\mathbf{w} \neq$ NULL do
13         $L \leftarrow L \setminus \{\mathbf{w}\}$
14         Remove $\mathbf{w}$ from all hash tables $\mathcal{T}_1, \dots, \mathcal{T}_t$
15         Reduce $\mathbf{w}$ with $\mathbf{v}$
16         if $\mathbf{w} \neq \mathbf{0}$ then
17             $S \leftarrow S \cup \{\mathbf{w}\}$
18     if $\mathbf{v}$ has changed and $\mathbf{v} \neq \mathbf{0}$ then
19         $S$.push($\mathbf{v}$)
20     if $\mathbf{v} = \mathbf{0}$ then
21         $K \leftarrow K + 1$
22     else
23         $L \leftarrow L \cup \{\mathbf{v}\}$
24         For each $i \in [t]$, add $\mathbf{v}$ to the bucket $\mathcal{T}_i[h_i(\mathbf{v})]$ in the hash table $\mathcal{T}_i$
25 return shortest vector $\mathbf{v}^*$ in $L$
```

Phase oracle $\mathcal{O}^{(2)}_{\text{gauss}}$. Once again, one QRAM_L call is used to load $\mathbf{w}_i$, after which D κ-bit hybrid multipliers are used to obtain all the D terms $(\mathbf{w}_i)_j v_j$, $j \in [D]$. These D terms are then summed up in depth $\lceil \log_2 D \rceil$ using $D-1$ κ-bit out-of-place adders. At this point, one of the registers is $|\mathbf{v} \cdot \mathbf{w}_i\rangle$. In order to check for the condition $|\mathbf{v} \cdot \mathbf{w}_i| \geq \|\mathbf{v}\|^2/2$, we can first compute the sum $\mathbf{v} \cdot \mathbf{w}_i - \|\mathbf{v}\|^2/2$ by using a κ-bit adder and copy its highest-order bit onto a qubit in the $|-\rangle$ state for a phase kickback. The adder generates an ancillary register containing $|\mathbf{v} \cdot \mathbf{w}_i - \|\mathbf{v}\|^2/2\rangle$. In order to check whether $-\mathbf{v} \cdot \mathbf{w}_i \geq \|\mathbf{v}\|^2/2 \iff \mathbf{v} \cdot \mathbf{w}_i - \|\mathbf{v}\|^2/2 < -\|\mathbf{v}\|^2$, we can apply a second κ-bit adder between the ancillary register $|\mathbf{v} \cdot \mathbf{w}_i - \|\mathbf{v}\|^2/2\rangle$ and the classical input $\|\mathbf{v}\|^2$. The highest-order bit of the result $|\mathbf{v} \cdot \mathbf{w}_i + \|\mathbf{v}\|^2/2\rangle$ is then flipped, since we are checking for a negative number, and copied onto the $|-\rangle$ ancilla for a phase kickback. This implements the phase $(-1)^{g_{\text{gauss}}(i)}$ as at most one condition $\mathbf{v} \cdot \mathbf{w}_i \geq \|\mathbf{v}\|^2/2$ or $-\mathbf{v} \cdot \mathbf{w}_i \geq \|\mathbf{v}\|^2/2$ can be true. After this, we uncompute all the arithmetic operations and QRAM call. The required submodules are summarised in Table 2.

7.2.3 Using LSH/LSF in the GaussSieve

Similarly to the `NVSieve`, LSH/LSF can be used in the `GaussSieve` as a filter to get a preliminary set of vectors to search among: instead of using a brute-force list search, we can only search through the candidate vectors C that hash to the same value (that is, they are close-by). The modified algorithm is given in Algorithm 4. The main idea is again to employ hash tables $\mathcal{T}_1, \dots, \mathcal{T}_t$ and replace the search over the entire list L with a shorter list of candidates $C = \bigcup_{i=1}^{t} \mathcal{T}_i[h_i(\mathbf{v})]$ that collide with the target vector $\mathbf{v}$ in at least one of the buckets $\mathcal{T}_1[h_1(\mathbf{v})], \dots, \mathcal{T}_t[h_t(\mathbf{v})]$. Once again, in order to use Grover's search, we must first classically gather all the indices of the vectors that collide with $\mathbf{v}$. If $C = \{\mathbf{w}_{j_1}, \mathbf{w}_{j_2}, \dots, \mathbf{w}_{j_{|C|}}\}$, then we use the indices $\{j_1, j_2, \dots, j_{|C|}\}$ to access the vectors in C via RAM calls and thus perform the classically controlled CNOTs in during a QRAM call. The phase $\mathcal{O}^{(1)}_{\text{gauss}} : |i\rangle \mapsto (-1)^{f_{\text{gauss}}(i)}|i\rangle$, where $f_{\text{gauss}} : [|C|] \to \{0,1\}$ is defined by $f_{\text{gauss}}(i) = 1$ if and only if either $\mathbf{w}_{j_i} \cdot (\mathbf{w}_{j_i} + 2\mathbf{v}) < 0$ or $\mathbf{w}_{j_i} \cdot (\mathbf{w}_{j_i} - 2\mathbf{v}) < 0$, is hence implemented first by one QRAM call and $|C|$ RAM calls, similarly to Equation (7), after which the remaining addition and multiplication operations explained in the previous section are performed (plus overall uncomputation). The exact same procedure is required in the phase oracle $\mathcal{O}^{(2)}_{\text{gauss}} : |i\rangle \mapsto (-1)^{g_{\text{gauss}}(i)}|i\rangle$, where $g_{\text{gauss}} : [|C|] \to \{0,1\}$ is defined by $g_{\text{gauss}}(i) = 1$ if and only if $|\mathbf{v} \cdot \mathbf{w}_{j_i}| \geq \|\mathbf{v}\|^2/2$. Both oracles $\mathcal{O}^{(1)}_{\text{gauss}}$ and $\mathcal{O}^{(2)}_{\text{gauss}}$ thus require one QRAM call of size $|C|$. Table 2 summarises the subroutines needed to implement both phase oracles.

7.2.4 GaussSieve with angular LSH

Hashing all vectors in the list L requires, similarly to `NVSieve`, $2k \cdot |L| \cdot t$ multiplications and additions, where $k = \log_{3/2} t - \log_{3/2} \ln(1/\varepsilon)$. The list of candidates on Line 8 has size $|C| \approx |L| \cdot p_2^*$, where p_2^* is given by Equation (1).

Classical complexity. The classical time spent searching over C is $O(D \cdot I \cdot |L| \cdot p_2^*)$, where I is the number of iterations of `GaussSieve`. To be more precise, the first search loop over C (Line 10) requires $4D - 2$ additions and $2D$ multiplications to check whether one vector can reduce another, while the second search loop over C (Line 12) requires $D+1$ additions and D multiplications (see Table 2). The number of hash tables t is determined by balancing the time hashing $O(k \cdot |L| \cdot t)$ with the time searching $O(D \cdot I \cdot |L| \cdot p_2^*)$. The asymptotic classical time and space complexities are $2^{0.336562D+o(D)}$ [127, 128]. We stress that the time complexities are only conjectures, in contrast to the `NVSieve`, where bounds can be proven under reasonable assumptions.

Quantum complexity. The quantum time spent searching over C is $O(D \cdot I\sqrt{|L| \cdot p_2^*})$. The number of hash tables t is determined by balancing the time hashing $O(k \cdot |L| \cdot t)$ with the time searching $O(D \cdot I\sqrt{|L| \cdot p_2^*})$. The asymptotic quantum time and space complexities are $2^{0.285949D+o(D)}$ [130, 128].

7.2.5 GaussSieve with spherical LSH

Hashing all vectors in the list L requires, similarly to `NVSieve`, $D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot |L|$ multiplications and additions, where $k = 6(\ln t - \ln\ln(1/\varepsilon))/\sqrt{D}$. The list of candidates on Line 8 has size $|C| \approx |L| \cdot p_2^*$, where p_2^* is given by Equation (3).

Classical complexity. Similarly to angular LSH, the first search loop over C (Line 10) requires $4D - 2$ additions and $2D$ multiplications to check whether one vector can reduce another, while the second search loop over C (Line 12) requires $D+1$ additions and D multiplications (see Table 2). The number of hash tables t is determined by balancing the time hashing $O(D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot |L|)$ with the time searching $O(D \cdot I \cdot |L| \cdot p_2^*)$. The asymptotic classical time and space complexities are $2^{0.297143D+o(D)}$ [127, 128].

Quantum complexity. The quantum time spent searching over C is $O(D \cdot I\sqrt{|L| \cdot p_2^*})$. The number of hash tables t is determined by balancing the time hashing $O(k \cdot |L| \cdot t)$ with the time searching $O(D \cdot I\sqrt{|L| \cdot p_2^*})$. The asymptotic quantum time and space complexities are $2^{0.267100D+o(D)}$ [130, 128].

7.2.6 GaussSieve with spherical LSF

We fix $k = 1$ concatenated filters per bucket and $t = \ln(1/\varepsilon)/\mathcal{W}_D(\alpha, \alpha, \pi/3)$ hash tables. Inserting all the vectors in L into relevant filters requires approximately $2\log_2 D \cdot |L| \cdot t \cdot \mathcal{C}_D(\alpha)$ additions. The list of candidates on Line 7 has size $|C| \approx |L| \cdot t \cdot \mathcal{C}_D(\alpha)^2$.

Classical complexity. Again, the first search loop over C requires $4D - 2$ additions and $2D$ multiplications to check whether one vector can reduce another, while the second search loop over C requires $D + 1$ additions and D multiplications. The parameter α is determined by minimising the sum of the time coming from filtering $O(\log D \cdot |L| \cdot t \cdot \mathcal{C}_D(\alpha))$ and the time coming from searching $O(D \cdot I \cdot |L| \cdot t \cdot \mathcal{C}_D(\alpha)^2)$. The asymptotic classical time and space complexities are $(3/2)^{D/2+o(D)} \approx 2^{0.292481D+o(D)}$ [32, 128].

Quantum complexity. The quantum time spent comparing vectors that collide on relevant filters is now $O(D \cdot I\sqrt{|L| \cdot t \cdot \mathcal{C}_D(\alpha)^2})$. The parameter α is determined by minimising the time required to filter plus the time required to search. The asymptotic quantum complexities are $(13/9)^{D/2+o(D)} \approx 2^{0.265257D+o(D)}$ [128].

8 Resource estimation analysis

In this section, we perform a thorough resource estimation required to implement Grover's search to speed-up the `NVSieve` and `GaussSieve` algorithms, both with and without LSH techniques. For such, we take into consideration the cost of arithmetic circuits from Section 4 and QRAM from Section 6 in implementing the phase oracles from Grover's search (Section 5), together with the overhead coming from quantum error correction and magic state distillation from Section 3. Our analysis will cover several facets from the quantum computation part within the sieving algorithms: circuit size and depth, number of logical and physical qubits, and overall runtime. Moreover, we shall analyse the most expensive sieving step and the total cost of *all* sieving steps (which includes smaller list sizes). We shall also gauge the impact of an error-corrected QRAM by suppressing its costs and comparing the end result with the full algorithmic cost. This shall be important from NIST's standpoint, because for the purpose of the standardization of post-quantum cryptographic technologies, it would be prudent to consider the possibility of a breakthrough quantum memory architecture making efficient queries possible. We start by describing how all the pieces from the previous sections fit together and the cost analysis is done in the case of lattice dimension $D = 400$, which is roughly the dimension in which SVP has to be solved to be able to break the minimally secure post-quantum cryptographic standards currently being standardised [27].

8.1 Case study: $D = 400$

8.1.1 NVSieve without LSH/LSF

Let us consider the case where the rank of the lattice is $D = 400$ and analyse the cost of employing Grover's search in the `NVSieve` without LSH/LSF from Algorithm 1. For simplicity, we will focus on one Grover's search. Even though the sizes of L and S are random, we assume a worst-case list of centers of size $|S| = 2^{0.2352D+0.102\log_2 D+2.45} \approx 2.15 \cdot 10^{29}$ and a list of size $|L| = D|S| \approx 8.61 \cdot 10^{31}$ as mentioned in Section 7.1.1. Moreover, we assume there is only one solution to each Grover's search.

Logical costs. The first step is to gather all the logical costs like Toffoli-count, number of logical qubits (circuit's width), and the circuit's active volume. According to Table 2, the phase oracle $\mathcal{O}_{\mathrm{NV}}$ from Grover's search requires 1 QRAM call, $2D$ κ-bit adders, and D κ-bit multipliers. Since the expected number of Grover iterations is $\lceil 7.67\sqrt{|S|}\rceil$ per Grover's search, we require

$$\mathsf{Toffoli\text{-}count}: \ \underbrace{\lceil 7.67\sqrt{|S|}\rceil}_{\text{Grover iterations}} \ \Big(\underbrace{|S|-2}_{\mathsf{QRAM}} + \underbrace{2D(\kappa-1)}_{2D \text{ adders}} + \underbrace{D(\kappa^2-\kappa+1)}_{D \text{ multipliers}} + \underbrace{\lceil \log_2|S|\rceil - 1}_{\text{Diffusion operator}}\Big) \approx 7.65\cdot 10^{44}.$$

Regarding the number of logical qubits, ancillae can be reused from one iteration to the next, so the maximum width (dirty ancillae plus input/output qubits) of Grover's circuit comes from QRAM plus the arithmetic operations and diffusion operator. One QRAM call needs $2|S| - \lceil\log_2|S|\rceil - 1$ dirty ancillae, plus $\lceil\log_2|S|\rceil + D\kappa$ qubits from input/output registers. On the other hand, the first D adders have a width of $3D\kappa$; the following D multipliers have a width of $D(2\kappa^2+\kappa)$; the subsequent $D-1$ adders have a width of $(2D-1)\kappa$; the final adder has a width of 3κ. Taking into account the overlap between different widths, since the output of one step is the input of the subsequent one, the total amount of logical qubits required is

$$\text{Logical qubits}: 2\Big(\underbrace{2|S| + D\kappa - 1}_{\mathsf{QRAM}} + \underbrace{2D\kappa}_{D\text{ adders}} + \underbrace{2D\kappa^2}_{D\text{ multipliers}} + \underbrace{D\kappa+\kappa}_{D\text{ adders}}\Big) \approx 8.61\cdot 10^{29},$$

where the factor 2 takes into account the space overhead coming from fast data blocks in baseline architectures, and from workspace qubits in active-volume architectures.

The active volume of the whole circuit is calculated by simply summing up the active volumes of 1 QRAM call, $2D$ κ-bit adders, and D κ-bit multipliers. Using the bucket-bridage QRAM from Lemma 9 and $C_{|CCZ\rangle} = 65$, the active volume of one Grover's search is

$$\text{Active volume}: \ \underbrace{\lceil 7.67\sqrt{|S|}\rceil}_{\text{Grover iterations}} \ \Big(\underbrace{(25+1.5\kappa + C_{|CCZ\rangle})|S|}_{\mathsf{QRAM}} + \underbrace{2D((\kappa-1)(39+C_{|CCZ\rangle})+7)}_{\text{Adders}}$$
$$+ \underbrace{D(28\kappa^2 - 42\kappa + 28 + (\kappa^2-\kappa+1)C_{|CCZ\rangle})}_{\text{Multipliers}} + \underbrace{(\lceil\log_2|S|\rceil - 1)(18 + C_{|CCZ\rangle})}_{\text{Diffusion operator}}\Big) \approx 1.06\cdot 10^{47}.$$

The reaction depth (which, in our case, is double the Toffoli-depth) follows from a simple concatenation of all the individual operations. The reaction depth of the phase oracle is the sum of reaction depths of one QRAM call, one κ-bit multiplier, and $2 + \lceil\log_2 D\rceil$ κ-bit adders. By adding the reaction depth of the diffusion operator (Fact 7) and multiplying the result by the number of Grover iterations $\lceil 7.67\sqrt{|S|}\rceil$, the get

$$\text{Reaction depth}: \ \underbrace{\lceil 7.67\sqrt{|S|}\rceil}_{\text{Grover iterations}} \ \Big(\underbrace{2\lceil\log_2|S|\rceil - 2}_{\mathsf{QRAM}} + \underbrace{2\kappa\log_2\kappa - 2\kappa - 2\log_2\kappa + 4}_{\text{Multipliers}}$$
$$+ \underbrace{2(\lceil\log_2 D\rceil + 2)(\kappa-1)}_{\text{Adders}} + \underbrace{2\lceil\log_2\lceil\log_2|S|\rceil\rceil}_{\text{Diffusion operator}}\Big) \approx 4.06\cdot 10^{18}.$$

Code distance and time. First consider a baseline architecture. Consider that there are enough distillation factories (see below) such that each Toffoli layer is performed every 4 logical cycles. Then one Grover's search employs $8.61\cdot 10^{29}$ logical qubits and $8.12\cdot 10^{18}$ logical cycles, to a total spacetime volume of $6.98\cdot 10^{48}$ logical blocks of size d^3. In order to keep a logical error probability below 0.1% per Grover's search, we must choose a code distance d such that

$$6.99\cdot 10^{48}\cdot d\cdot 0.1(100p_{\text{phy}})^{(d+1)/2} \leq 0.001.$$

With physical error $p_{\text{phy}} = 10^{-5}$, the above is satisfied by $d = 34$, which yields a logical error probability of $\approx 0.08\%$. Since each logical qubit requires $2d^2$ physical qubits (taking into account the ancillae required for the check operators measurements), one Grover's search employs $1.99\cdot 10^{33}$ physical qubits. With a code cycle of 100 ns, the circuit time of one Grover's search is $8.75\cdot 10^5$ years.

Consider now an active-volume architecture. With $8.61 \cdot 10^{29}$ logical qubits and an active volume of $1.06 \cdot 10^{47}$, the total spacetime volume is $2.11 \cdot 10^{47}$ logical blocks of size d^3, twice the active volume. The number of logical cycles is $2(1.06 \cdot 10^{47})/(8.61 \cdot 10^{29}) = 2.45 \cdot 10^{17}$ per Grover's search, since only half the logical qubits, the workspace qubits, execute logical blocks in every logical cycle. In order to keep a logical error probability below 0.1%, we must choose a code distance d such that

$$2.11 \cdot 10^{47} \cdot d \cdot 0.1(100 p_{\text{phy}})^{(d+1)/2} \leq 0.001.$$

With physical error $p_{\text{phy}} = 10^{-5}$, the above is satisfied by $d = 34$, which yields a logical error probability of $\approx 0.002\%$. Since each logical qubit requires d^2 physical qubits, one Grover's search employs $9.95 \cdot 10^{32}$ physical qubits. With a code cycle of 100 ns, the circuit time of one Grover's search is $\approx 6,620$ years.

Due to the sequential natural of classical processing associated with surface-code-based quantum computation, the runtime of every circuit is limited by its reaction depth. Given the reaction depth of $4.06 \cdot 10^{18}$ and a reaction time of 1 μs, the Grover's search is thus reaction limited at $\approx 1.29 \cdot 10^5$ years. This limits the active-volume architecture to a runtime of $1.29 \cdot 10^5$ years, and not $\approx 6,620$ years.

Distillation protocol. Finally, we determine the distillation protocol necessary for the computation, which is obtained from the Toffoli-count. We require that the error probability of performing $7.65 \cdot 10^{44}$ Toffoli gates be less than 0.1%, which means that each magic state $|CCZ\rangle$ must have an error rate below $1.31 \cdot 10^{-48}$. For baseline architectures with the above code distance $d = 34$, the distillation protocol $(\text{15-to-1})^4_{\lceil d/4\rceil,\lceil d/8\rceil,\lceil d/8\rceil} \times (\text{15-to-1})^4_{\lceil d/2\rceil,\lceil d/4\rceil,\lceil d/4\rceil} \times (\text{8-to-CCZ})_{d,\lceil d/2\rceil,\lceil d/2\rceil}$ outputs a magic state $|CCZ\rangle$ with error rate of $5.4 \cdot 10^{-50}$ every 108 code cycles by using 111,192 physical qubits, which is enough for our needs. Since each Toffoli layer must be executed every $4d = 132$ code cycles, we require $\approx \frac{108}{132}|S|/2 = 8.54 \cdot 10^{28}$ distillation factories running in parallel, which adds another $9.50 \cdot 10^{33}$ physical qubits to a total of $1.15 \cdot 10^{34}$ physical qubits. Regarding active-volume architectures, on the other hand, the same $(\text{15-to-1})^4_{\lceil d/4\rceil,\lceil d/8\rceil,\lceil d/8\rceil} \times (\text{15-to-1})^4_{\lceil d/2\rceil,\lceil d/4\rceil,\lceil d/4\rceil} \times (\text{8-to-CCZ})_{d,\lceil d/2\rceil,\lceil d/2\rceil}$ protocol with $d = 34$ outputs a magic state $|CCZ\rangle$ with error rate of $5.4 \cdot 10^{-50}$. The associated resources are already included in the active volume cost $C_{|CCZ\rangle}$.

8.1.2 GaussSieve without LSH/LSF

We move on to analysing the cost of Grover's search in the `GaussSieve` without LSH/LSF (Algorithm 3). The analysis of `GaussSieve` is harder since it is a heuristic algorithm with few proven properties. In each sieving step, there are two search loops that are called while a solution can be found. For simplicity, we consider one Grover's search with $M = 1$ solution. Another heuristic parameter of the algorithm is the list size. Here we assume a sieving step with list size $|L| = 2^{0.193D+2.325} \approx 8.70 \cdot 10^{23}$ as reported by us (see Section 7.2.1).

Logical costs. Once again we gather all the logical costs first. In each sieving steps, there are two different search loops being performed. According to Table 2, the phase oracle $\mathcal{O}^{(1)}_{\text{gauss}}$ from the first loop requires 1 QRAM call of size $|L|$, $4D - 2$ κ-bit adders, and $2D$ κ-bit multipliers, while the phase oracle $\mathcal{O}^{(2)}_{\text{gauss}}$ from the second loop requires 1 QRAM call of size $|L|$, $D + 1$ κ-bit adders, and D κ-bit hybrid multipliers. The expected number of Grover iterations is $\lceil 7.67\sqrt{|L|}\rceil$. The Toffoli-count of the two search loops is

$$\text{Toffoli-count loop 1}: \underbrace{\lceil 7.67\sqrt{|L|}\rceil}_{\text{Iterations}}\Big(\underbrace{|L|-2}_{\text{QRAM}} + \underbrace{(4D-2)(\kappa-1)}_{4D-2\text{ adders}} + \underbrace{2D(\kappa^2-\kappa+1)}_{2D\text{ multipliers}} + \underbrace{\lceil\log_2|L|\rceil - 1}_{\text{Difussion operator}}\Big),$$

$$\text{Toffoli-count loop 2}: \underbrace{\lceil 7.67\sqrt{|L|}\rceil}_{\text{Iterations}}\Big(\underbrace{|L|-2}_{\text{QRAM}} + \underbrace{(D+1)(\kappa-1)}_{5D-1\text{ adders}} + \underbrace{D(0.5\kappa^2-1.5\kappa+1)}_{D\text{ multipliers}} + \underbrace{\lceil\log_2|L|\rceil - 1}_{\text{Difussion operator}}\Big),$$

both approximately equal to $6.22 \cdot 10^{36}$.

Table 3: Summary of required resources to perform one Grover's search with one solution in the `NVSieve` with and without LSH/LSF assuming baseline and active-volume physical architectures. Reaction limit and circuit time are measured in days, and final time is the maximum between both. We assume a lattice dimension $D = 400$, topological and magic distillation probability errors smaller than 10^{-3}, and a Grover's search probability error of 10^{-3}. `NVSieve` without LSH has list of centers of size $|S| = 2^{0.2352D+0.102\log_2 D+2.45}$. `NVSieve` with LSH/LSF replaces S with a list of candidates of size $|C| = |S| \cdot p_2^*$ for LSH and $|C| = |S| \cdot \mathcal{C}_D(\alpha)^2 \cdot \ln(1/\varepsilon)/\mathcal{W}_D(\alpha, \alpha, \pi/3)$ for LSF, where $\varepsilon = 10^{-3}$.

	Resource/Sieve	`NVSieve`	`NVSieve` + angular LSH	`NVSieve` + spherical LSH	`NVSieve` + spherical LSF
	List size	$2.15 \cdot 10^{29}$	$3.46 \cdot 10^{21}$	$2.71 \cdot 10^{20}$	$1.35 \cdot 10^{15}$
	Hashing parameter k	-	83	5	1
	Number hash tables t	-	$2.28 \cdot 10^{15}$	$2.75 \cdot 10^{7}$	$2.84 \cdot 10^{38}$
	Filter angle α	-	-	-	$\pi/3$
	Logical qubits	$8.61 \cdot 10^{29}$	$1.39 \cdot 10^{22}$	$1.08 \cdot 10^{21}$	$5.42 \cdot 10^{15}$
	Toffoli-count	$7.65 \cdot 10^{44}$	$1.56 \cdot 10^{33}$	$3.42 \cdot 10^{31}$	$3.82 \cdot 10^{23}$
	Toffoli-width	$1.08 \cdot 10^{29}$	$1.73 \cdot 10^{21}$	$1.35 \cdot 10^{20}$	$6.77 \cdot 10^{14}$
	Active volume	$1.06 \cdot 10^{47}$	$2.16 \cdot 10^{35}$	$4.71 \cdot 10^{33}$	$5.28 \cdot 10^{25}$
	Reaction depth	$4.06 \cdot 10^{18}$	$4.91 \cdot 10^{14}$	$1.36 \cdot 10^{14}$	$2.95 \cdot 10^{11}$
	Reaction limit (days)	$1.13 \cdot 10^{9}$	$1.36 \cdot 10^{5}$	$3.79 \cdot 10^{4}$	$8.19 \cdot 10^{1}$
Baseline	Code distance	34	27	25	20
	Distillation factories	$8.54 \cdot 10^{28}$	$1.35 \cdot 10^{21}$	$1.14 \cdot 10^{20}$	$5.08 \cdot 10^{14}$
	Physical qubits	$1.15 \cdot 10^{34}$	$1.16 \cdot 10^{26}$	$8.78 \cdot 10^{24}$	$2.33 \cdot 10^{19}$
	Circuit time (days)	$7.66 \cdot 10^{9}$	$7.37 \cdot 10^{5}$	$1.89 \cdot 10^{5}$	$3.27 \cdot 10^{2}$
	Final time (days)	$7.66 \cdot 10^{9}$	$7.37 \cdot 10^{5}$	$1.89 \cdot 10^{5}$	$3.27 \cdot 10^{2}$
Active-volume	Code distance	34	26	24	20
	Physical qubits	$9.95 \cdot 10^{32}$	$9.37 \cdot 10^{24}$	$6.24 \cdot 10^{23}$	$2.17 \cdot 10^{18}$
	Circuit time (days)	$5.80 \cdot 10^{7}$	$5.62 \cdot 10^{3}$	$1.45 \cdot 10^{3}$	$2.71 \cdot 10^{0}$
	Final time (days)	$1.13 \cdot 10^{9}$	$1.36 \cdot 10^{5}$	$3.79 \cdot 10^{4}$	$8.19 \cdot 10^{1}$

Regarding the number of logical qubits, the first search loop requires (already taking overlaps into account) $2|L| + \kappa D - 1$ qubits for the QRAM, $D\kappa$ qubits after copying $|\mathbf{w}_i\rangle$ once, $4D\kappa$ qubits for the parallel $2D$ κ-bit adders, $2D(2\kappa^2 - \kappa)$ qubits for the parallel $2D$ κ-bit multipliers, and finally $2\kappa(D-1)$ qubits for the final $2D - 2$ κ-bit adders. The second search loop requires (already taking overlaps into account) $2|L| + \kappa D - 1$ qubits for the QRAM, $D(1.5\kappa^2 - 0.5\kappa)$ qubits for the D κ-bit hybrid multipliers, $(D-1)\kappa$ qubits for the $D - 1$ κ-bit adders, and finally 3κ qubits for the last 2 κ-bit adders. It is not hard to see that the first search loop employs the most logical qubits, which is the final count:

$$\text{Logical qubits}: 2\Big(\underbrace{2|L| + D\kappa - 1}_{\mathsf{QRAM}} + \underbrace{D\kappa}_{\text{Copying}} + \underbrace{4D\kappa}_{2D\text{ adders}} + \underbrace{2D(2\kappa^2 - \kappa)}_{2D\text{ multipliers}} + \underbrace{2(D-1)\kappa}_{2D-2\text{ adders}}\Big) \approx 3.48 \cdot 10^{24}.$$

The active volume of both search loops is simply the sum of the individual active volumes,

$$\text{Active volume loop 1}: \underbrace{\lceil 7.67\sqrt{|L|}\rceil}_{\text{Iterations}} \Big(\underbrace{(25 + 1.5\kappa + C_{|CCZ\rangle})|L|}_{\mathsf{QRAM}} + \underbrace{(\lceil \log_2 |L| \rceil - 1)(18 + C_{|CCZ\rangle})}_{\text{Diffusion operator}}$$
$$+ \underbrace{(4D-2)((\kappa - 1)(39 + C_{|CCZ\rangle}) + 7)}_{\text{Adders}} + \underbrace{4(2D\kappa + 4)}_{\text{Extra }\mathsf{CNOT}\text{s}}$$

$$+ \underbrace{2D(28\kappa^2 - 42\kappa + 28 + (\kappa^2 - \kappa + 1)C_{|CCZ\rangle})}_{\text{Multipliers}}\Big),$$

$$\text{Active volume loop 2}: \underbrace{\lceil 7.67\sqrt{|L|}\rceil}_{\text{Iterations}}\Big(\underbrace{(25 + 1.5\kappa + C_{|CCZ\rangle})|L|}_{\text{QRAM}} + \underbrace{(\lceil \log_2 |L|\rceil - 1)(18 + C_{|CCZ\rangle})}_{\text{Diffusion operator}}$$
$$+ \underbrace{(D+1)((\kappa - 1)(39 + C_{|CCZ\rangle}) + 7)}_{\text{Adders}}$$
$$+ \underbrace{D(20.25\kappa^2 - 48.75\kappa + 32 + (0.5\kappa^2 - 1.5\kappa + 1)C_{|CCZ\rangle})}_{\text{Multipliers}}\Big),$$

both approximately equal to $8.59 \cdot 10^{38}$, while the reaction depth of one Grover's search in each search loop is

$$\text{Reaction depth}: \underbrace{\lceil 7.67\sqrt{|L|}\rceil}_{\text{Iterations}}\Big(\underbrace{2\lceil \log_2 |L|\rceil - 2}_{\text{QRAM}} + \underbrace{2(1 + \lceil \log_2 D\rceil)(\kappa - 1)}_{\text{Adders}} + \underbrace{2\kappa \log_2 \kappa - 2\kappa - 2\log_2 \kappa + 4}_{\text{Multipliers}}$$
$$+ \underbrace{2\lceil \log_2 \lceil \log_2 |L|\rceil\rceil}_{\text{Diffusion operator}}\Big) \approx 7.45 \cdot 10^{15}.$$

Code distance and time. The analysis is the same to the `NVSieve` case. First consider a baseline architecture and the Grover's search with $Q = \lceil 7.67\sqrt{|L|}\rceil$ iterations from the first search loop. Assuming enough distillation factories, each Toffoli layer is performed every 4 logical cycles. The Grover's search employs a total of $3.48 \cdot 10^{24}$ logical qubits and $1.49 \cdot 10^{16}$ logical cycles. In order to keep a logical error probability below 0.1%, we choose a code distance d such that

$$3.48 \cdot 10^{24} \cdot 1.49 \cdot 10^{16} \cdot d \cdot 0.1(100p_{\text{phy}})^{(d+1)/2} \leq 0.001.$$

Give $p_{\text{phy}} = 10^{-5}$, the above is satisfied by $d = 29$, yielding a logical error probability of $\approx 0.015\%$. With each logical qubit requiring $2d^2$ physical qubits, $5.85 \cdot 10^{27}$ physical qubits are used (excluding distillation qubits). With a code cycle of 100 ns, the Grover's circuit time is $\approx 1,370$ years. The same steps can be repeated for the other search loop, which we omit here.

Now consider an active-volume architecture. The Grover's search with $Q = \lceil 7.67\sqrt{|L|}\rceil$ iterations from the first search loop requires $3.48 \cdot 10^{24}$ logical qubits and active volume of $8.59 \cdot 10^{38}$, and therefore $2(8.59 \cdot 10^{38})/(3.48 \cdot 10^{24}) = 1.49 \cdot 10^{16}$ logical cycles. The code distance d is chosen so that

$$2 \cdot 8.59 \cdot 10^{38} \cdot d \cdot 0.1(100p_{\text{phy}})^{(d+1)/2} \leq 0.001.$$

Given $p_{\text{phy}} = 10^{-5}$, the above is satisfied by $d = 28$, yielding a logical error probability of $\approx 0.015\%$. With each logical qubit requiring d^2 physical qubits, $2.73 \cdot 10^{27}$ physical qubits are required. With a code cycle of 100 ns, the Grover's circuit time is ≈ 11 years. The same steps can be repeated for the other search loop, which we omit here.

Finally, given a reaction depth of $7.45 \cdot 10^{15}$ and a reaction time of 1 μs, the Grover's search is thus reaction limited at ≈ 230 years. This limits the active-volume execution time to ≈ 230 years, and not ≈ 11 years.

Distillation protocol. Finally, we check whether the distillation protocol $(\text{15-to-1})^4_{\lceil d/4\rceil,\lceil d/8\rceil,\lceil d/8\rceil} \times (\text{15-to-1})^4_{\lceil d/2\rceil,\lceil d/4\rceil,\lceil d/4\rceil} \times (\text{8-to-CCZ})_{d,\lceil d/2\rceil,\lceil d/2\rceil}$ with code distance $d = 29$ outputs magic states with error probability smaller than $0.001/(6.22 \cdot 10^{36}) = 1.61 \cdot 10^{-40}$. Indeed, the distillation protocol outputs magic states with error rate $1.0 \cdot 10^{-43}$ every 96 code cycles using 84,308 physical qubits. Since each Toffoli layer must be executed every $4d = 116$ code cycles, we require $\frac{96}{116}|L|/2 = 3.60 \cdot 10^{23}$ distillation factories, which adds another $3.03 \cdot 10^{28}$ physical qubits to a total of $3.62 \cdot 10^{28}$ physical qubits. For active-volume architectures, the distillation cost was already computed in $C_{|CCZ\rangle}$.

Table 4: Summary of required resources to perform one Grover's search with one solution in GaussSieve with and without LSH/LSF assuming baseline and active-volume physical architectures. Reaction limit and circuit time are measured in days, and final time is the maximum between both. We assume a lattice dimension $D = 400$, topological and magic distillation probability errors smaller than 10^{-3}, and a Grover's search probability error of 10^{-3}. We focus on a Grover's search from the first loop search. GaussSieve has list size $|L| = 2^{0.193D+2.325}$ and list of candidates of size $|C| = |L| \cdot p_2^*$ for LSH and $|C| = |L| \cdot \mathcal{C}_D(\alpha)^2 \cdot \ln(1/\varepsilon)/\mathcal{W}_D(\alpha, \alpha, \pi/3)$ for LSF, where $\varepsilon = 10^{-3}$.

	Resource/Sieve	GaussSieve	GaussSieve + angular LSH	GaussSieve + spherical LSH	GaussSieve + spherical LSF
	List size	$8.70 \cdot 10^{23}$	$5.00 \cdot 10^{14}$	$3.90 \cdot 10^{12}$	$5.48 \cdot 10^{9}$
	Hashing parameter k	-	99	7	1
	Number hash tables t	-	$1.57 \cdot 10^{18}$	$5.31 \cdot 10^{9}$	$2.84 \cdot 10^{38}$
	Filter angle α	-	-	-	$\pi/3$
	Logical qubits	$3.48 \cdot 10^{24}$	$2.00 \cdot 10^{15}$	$1.56 \cdot 10^{13}$	$2.19 \cdot 10^{10}$
	Toffoli-count	$6.22 \cdot 10^{36}$	$8.58 \cdot 10^{22}$	$5.91 \cdot 10^{19}$	$3.11 \cdot 10^{15}$
	Toffoli-width	$4.35 \cdot 10^{23}$	$2.50 \cdot 10^{14}$	$1.95 \cdot 10^{12}$	$2.74 \cdot 10^{9}$
	Active volume	$8.59 \cdot 10^{38}$	$1.18 \cdot 10^{25}$	$8.15 \cdot 10^{21}$	$4.29 \cdot 10^{17}$
	Reaction depth	$7.45 \cdot 10^{15}$	$1.68 \cdot 10^{11}$	$1.46 \cdot 10^{10}$	$5.37 \cdot 10^{8}$
	Reaction limit (hours)	$8.63 \cdot 10^{4}$	$4.66 \cdot 10^{1}$	$4.06 \cdot 10^{0}$	$1.49 \cdot 10^{-1}$
Baseline	Code distance	29	20	17	15
	Distillation factories	$3.60 \cdot 10^{23}$	$1.88 \cdot 10^{14}$	$1.72 \cdot 10^{12}$	$2.19 \cdot 10^{9}$
	Physical qubits	$3.62 \cdot 10^{28}$	$8.60 \cdot 10^{18}$	$6.47 \cdot 10^{16}$	$5.92 \cdot 10^{13}$
	Circuit time (hours)	$5.00 \cdot 10^{5}$	$1.86 \cdot 10^{2}$	$1.38 \cdot 10^{1}$	$4.47 \cdot 10^{-1}$
	Final time (hours)	$5.00 \cdot 10^{5}$	$1.86 \cdot 10^{2}$	$1.38 \cdot 10^{1}$	$4.47 \cdot 10^{-1}$
Active-volume	Code distance	28	20	16	14
	Physical qubits	$2.73 \cdot 10^{27}$	$8.00 \cdot 10^{17}$	$3.99 \cdot 10^{15}$	$4.29 \cdot 10^{12}$
	Circuit time (hours)	$4.00 \cdot 10^{3}$	$1.64 \cdot 10^{0}$	$1.16 \cdot 10^{-1}$	$3.81 \cdot 10^{-3}$
	Final time (hours)	$8.63 \cdot 10^{4}$	$4.66 \cdot 10^{1}$	$4.06 \cdot 10^{0}$	$1.49 \cdot 10^{-1}$

8.1.3 NVSieve and GaussSieve with LSH/LSF

Finally, we consider both NVSieve and GaussSieve with LSH/LSF. Once again, we focus on one Grover's search with worst-case list size $|S| = 2^{0.2352D+0.102\log_2 D+2.45} \approx 2.15 \cdot 10^{29}$ for NVSieve and $|L| = 2^{0.193D+2.325} \approx 8.70 \cdot 10^{23}$ for GaussSieve. We assume the existence of only one solution in each Grover's search, and we focus on the first search loop in GaussSieve. The average size of the list of candidates C to be searched over with Grover's algorithm is $|C| = |L| \cdot p_2^*$ for LSH and $|C| = |L| \cdot t \cdot \mathcal{C}_D(\alpha)^2$ for LSF.

The choice for the hashing parameter k and the number of hash tables t (and the angle α for LSF) is highly heuristic. For LSH the choice of k is usually based on guaranteeing that nearby vectors collide with high probability in at least one hash table. This yields $k = \log_{3/2} t - \log_{3/2} \ln(1/\varepsilon)$ for angular LSH and $k = 6(\ln t - \ln\ln(1/\varepsilon))/\sqrt{D}$ for spherical LSH. For spherical LSF, $k = 1$ and $t = \ln(1/\varepsilon)/\mathcal{W}_D(\alpha, \alpha, \pi/3)$. Here $\varepsilon = 10^{-3}$. On the other hand, the value of t for LSH is based on balancing the classical hashing time with the quantum searching time, while the parameter α is obtained by minimising the total runtime (classical hashing time plus quantum searching time). A precise choice for t and α thus depends on all sieving steps and not just a single Grover's search. We refer the reader to Section 8.2 below for a list of assumptions on the performance of NVSieve and GaussSieve that allows for precise expressions used to derive t and α. For now we just quote the values k, t, and α in Tables 3 and 4.

The analysis is very similar to the previous ones, so we omit most of the details and list the results in Tables 3 and 4. The expressions for Toffoli-count, number of logical qubits, active volume, and reaction depth are basically the aforementioned ones but replacing $|L|$ or $|S|$ with $|C|$ within a Grover's search.

Tables 3 and 4 show a rough estimate for one Grover's search with worst-case list size in each `NVSieve` and `GaussSieve` with and without LSH in dimension $D = 400$. Even though only one Grover's search was taken into consideration, we can already grasp the order of magnitude of each resource, specially number of physical qubits and overall time, the most important ones. Moreover, it is possible to observe some of the advantages and disadvantages of each algorithm, e.g., the use of hashing has a significant impact on time and number of physical qubits as expected from searching a smaller list. However, a full and complete analysis can only come from considering all Grover's searches from all sieving steps, which we shall look at next.

8.2 Resource estimations via heuristic assumptions

In this section, we employ the analysis procedure outlined above in order to gauge the required resources to fully carry out the `NVSieve` and `GaussSieve` aided by Grover's search. For the sake of comparison, we also consider a completely classical implementation where vector reductions are searched sequentially. Since these sieving algorithms involve several quantities which are difficult to precisely measure, we rely on heuristic and numerical observations from Sections 7.1.1 and 7.2.1 to build plausible worst-case assumptions on which the resource estimations can be performed. In the following, we assume that:

1. The initial list size $|L|$ in `NVSieve` is $|L| = D \cdot 2^{0.2352D+0.102\log_2 D+2.45}$.

2. In the classical implementation of `NVSieve`, the list S or C is scanned one full time in order to find a solution.

3. In the quantum implementation of `NVSieve`, there is only one solution to each Grover's search.

4. The center list has size $|S| = 2^{0.2352D+0.102\log_2 D+2.45}$ in each sieving step of `NVSieve` without LSH/LSF. The list size $|L|$ decreases by $|S|$ per sieving step.

5. In each sieving step of `NVSieve` with LSH, $|S| = 2^{0.2352D+0.102\log_2 D+2.45}$ vectors are inserted into t hash tables, and the list of candidates has size $|C| = |S| \cdot p_2^*$, where p_2^* is the average probability that far-away vectors collide. In `NVSieve` with LSF, $|S| = 2^{0.2352D+0.102\log_2 D+2.45}$ vectors are inserted into relevant filters out of t buckets, and the list of candidates has size $|C| = |S| \cdot t \cdot \mathcal{C}_D(\alpha)^2 = |S| \cdot \mathcal{C}_D(\alpha)^2 \cdot \ln(1/\varepsilon)/\mathcal{W}_D(\alpha, \alpha, \pi/3)$, where $\varepsilon = 10^{-3}$. The list size $|L|$ decreases by $|S|$ per sieving step.

6. The maximum list size in `GaussSieve` is $2^{0.193D+2.325}$, while the number of iterations I grows as $2^{0.283D+0.335}$.

7. The list size $|L|$ in `GaussSieve` equals the maximum list size of $2^{0.193D+2.325}$ for all iterations and its size therefore does not decrease.

8. In the classical implementation of `GaussSieve`, the list L or C in the first search loop (Line 7 in Algorithm 3 and Line 10 in Algorithm 4) is scanned 10 times: one vector reduction happens after every scan until no solutions are left after the 9-th time. The list L or C in the second search loop (Line 9 in Algorithm 3 and Line 12 in Algorithm 4) is scanned only once.

9. In the quantum implementation of `GaussSieve`, the first search loop (Line 7 in Algorithm 3 and Line 10 in Algorithm 4) is performed 10 times: 9 times with $M = 1$ solution, and 1 final time with $M = 0$ solutions. The second search loop (Line 9 in Algorithm 3 and Line 12 in Algorithm 4) is performed only once with $M = 0$ solutions.

Table 5: Amount of classical arithmetic operations in the *classical* implementation of `NVSieve` and `GaussSieve` with and without LSH/LSF. Here $|L|$ is the maximum list size of `NVSieve` or `GaussSieve`, I is the number of iterations of `GaussSieve`, and p_2^* is the average probability that a non-reducing vector collides with another vector in at least one of t hash tables.

	Searching		Hashing	
Sieve/Operations	Additions	Multiplications	Additions	Multiplications
`NVSieve`	$D \cdot \lvert L\rvert^2$	$\frac{1}{2}D^2 \cdot \lvert L\rvert^2$	0	0
`NVSieve` + angular LSH	$D \cdot \lvert L\rvert^2 \cdot p_2^*$	$\frac{1}{2}D \cdot \lvert L\rvert^2 \cdot p_2^*$	$k \cdot t \cdot \lvert L\rvert$	$2k \cdot t \cdot \lvert L\rvert$
`NVSieve` + spherical LSH	$D \cdot \lvert L\rvert^2 \cdot p_2^*$	$\frac{1}{2}D \cdot \lvert L\rvert^2 \cdot p_2^*$	$D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot \lvert L\rvert$	$D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot \lvert L\rvert$
`NVSieve` + spherical LSF	$D \cdot \lvert L\rvert^2 \cdot t \cdot \mathcal{C}_D(\alpha)^2$	$\frac{1}{2}D \cdot \lvert L\rvert^2 \cdot t \cdot \mathcal{C}_D(\alpha)^2$	$2\log_2 D \cdot \lvert L\rvert \cdot t \cdot \mathcal{C}_D(\alpha)$	0
`GaussSieve`	$(41D-19) \cdot I \cdot \lvert L\rvert$	$21D \cdot I \cdot \lvert L\rvert$	0	0
`GaussSieve` + angular LSH	$(41D-19) \cdot I \cdot \lvert L\rvert \cdot p_2^*$	$21D \cdot I \cdot \lvert L\rvert \cdot p_2^*$	$k \cdot t \cdot \lvert L\rvert$	$2k \cdot t \cdot \lvert L\rvert$
`GaussSieve` + spherical LSH	$(41D-19) \cdot I \cdot \lvert L\rvert \cdot p_2^*$	$21D \cdot I \cdot \lvert L\rvert \cdot p_2^*$	$D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot \lvert L\rvert$	$D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot \lvert L\rvert$
`GaussSieve` + spherical LSF	$(41D-19) \cdot I \cdot \lvert L\rvert \cdot t \cdot \mathcal{C}_D(\alpha)^2$	$21D \cdot I \cdot \lvert L\rvert \cdot t \cdot \mathcal{C}_D(\alpha)^2$	$2\log_2 D \cdot \lvert L\rvert \cdot t \cdot \mathcal{C}_D(\alpha)$	0

10. In `GaussSieve` with LSH, $|L| = 2^{0.193D+2.325}$ vectors are inserted into t hash tables and the list of candidates has size $|C| = |L| \cdot p_2^*$, where p_2^* is the average probability that far-away vectors collide. In `GaussSieve` with LSF, $|L| = 2^{0.193D+2.325}$ vectors are inserted into relevant filters out of t buckets and the list of candidates has size $|C| = |L| \cdot t \cdot \mathcal{C}_D(\alpha)^2$.

11. Angular LSH: $k = \log_{3/2} t - \log_{3/2} \ln(1/\varepsilon)$ and the average collision probability of far-away vectors p_2^* is given by Equation (1). The total classical hashing time requires $2k \cdot t \cdot |L|$ multiplications and $k \cdot t \cdot |L|$ additions. In the quantum implementations, the number of hash tables is determined through the equality $D^2 \cdot |L|\sqrt{|S| \cdot p_2^*} = k \cdot t \cdot |L|$ for `NVSieve` (there are $|L|/|S| = D$ sieving steps) and $D \cdot I\sqrt{|L| \cdot p_2^*} = k \cdot t \cdot |L|$ for `GaussSieve`.

12. Spherical LSH: $k = 6(\ln t - \ln\ln(1/\varepsilon))/\sqrt{D}$ and the average collision probability of far-away vectors p_2^* is given by Equation (3). The total classical hashing time requires $D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot |L|$ additions and multiplications. In the quantum implementations, the number of hash tables is determined through the equality $D^2 \cdot |L|\sqrt{|S| \cdot p_2^*} = D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot |L|$ for `NVSieve` (there are $|L|/|S| = D$ sieving steps) and $D \cdot I\sqrt{|L| \cdot p_2^*} = D \cdot 2^{\sqrt{D}} \cdot k \cdot t \cdot |L|$ for `GaussSieve`.

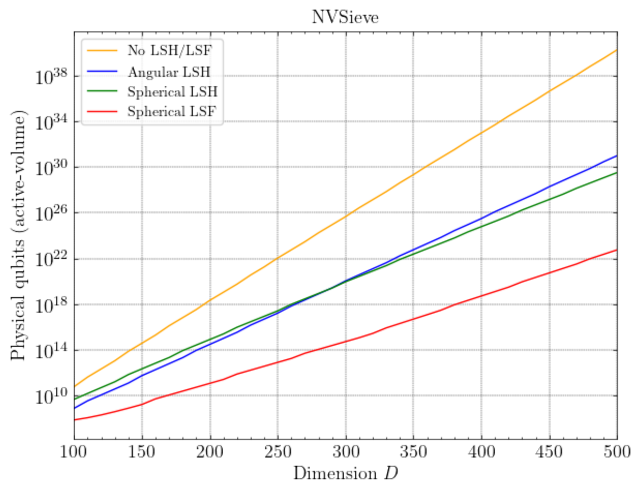
13. Spherical LSF: $k = 1$ and the number of filter buckets is $t = \ln(1/\varepsilon)/\mathcal{W}_D(\alpha, \alpha, \pi/3)$ with $\varepsilon = 10^{-3}$. The total classical time to place vectors into relevant filters requires $2\log_2 D \cdot |L| \cdot t \cdot \mathcal{C}_D(\alpha)$ additions. In the quantum implementations, the parameter $\alpha \leq 1/2$ is determined by minimising $\log_2 D \cdot |L| \cdot t \cdot \mathcal{C}_D(\alpha) + D^2 \cdot |L|\sqrt{|S| \cdot t \cdot \mathcal{C}_D(\alpha)^2}$ for `NVSieve` and $\log_2 D \cdot |L| \cdot t \cdot \mathcal{C}_D(\alpha) + D \cdot I\sqrt{|L| \cdot t \cdot \mathcal{C}_D(\alpha)^2}$ for `GaussSieve`.

14. Classical additions and multiplications require 1 and 4 computational cycles, respectively.

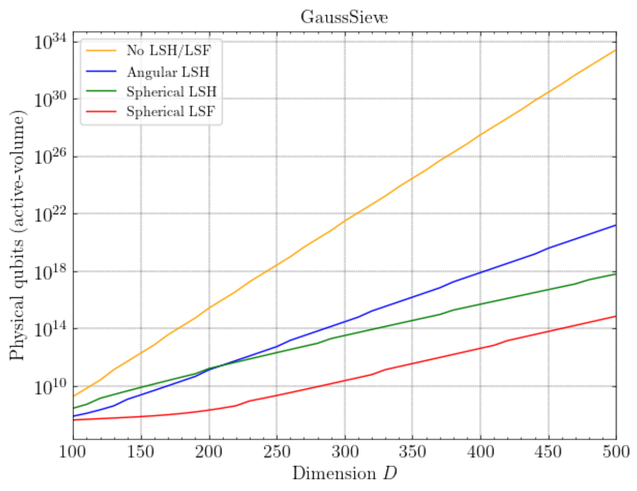
15. The topological error probability and Grover's search error probability (δ in Fact 7) are $\leq 10^{-3}$.

For convenience, under the above assumptions, in Table 5 we collect all classical operations coming from hashing and searching for the classical implementation of the sieving algorithms.

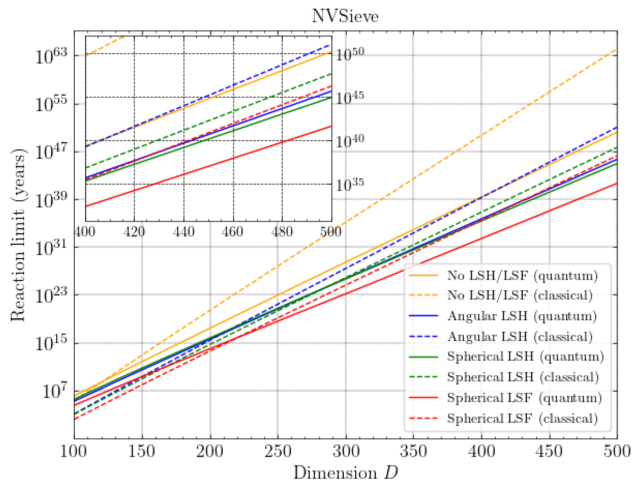
In Figure 5 we compare the number of physical qubits and reaction limits from `NVSieve` and `GaussSieve` with and without LSH/LSF under an active-volume architecture. The estimated classical execution times using a 6-GHz-clock-speed single-core classical computer are also included, where GHz means 10^9 operations per second. For completeness, we also add the classical hashing time to the reaction limits coming from the Grover's search, although the difference is tiny. The use of locality-sensitive techniques greatly improves both quantities, specially the amount of physical qubits. It



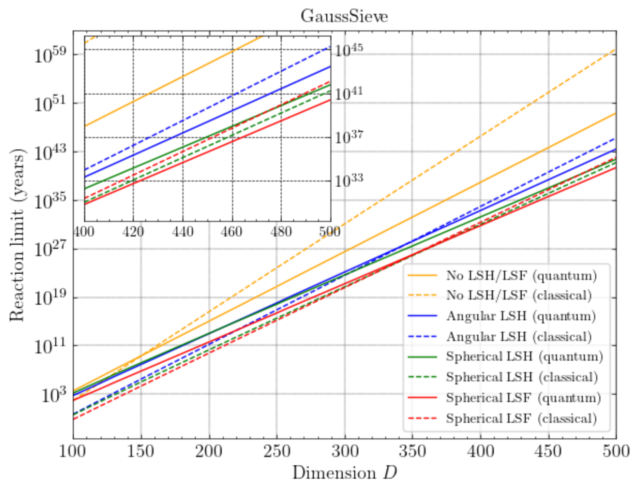
(a) Active-volume physical qubits of `NVSieve`



(b) Active-volume physical qubits of `GaussSieve`



(c) Reaction limit of `NVSieve`

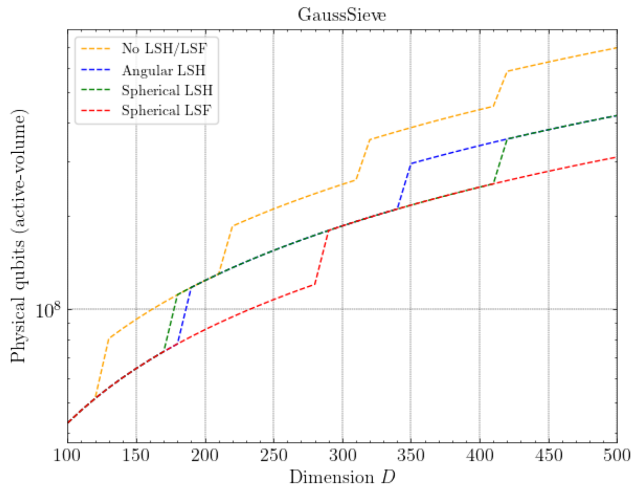


(d) Reaction limit of `GaussSieve`

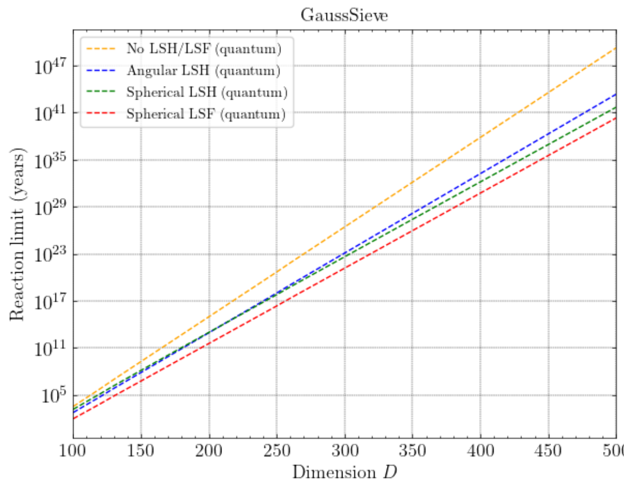
Figure 5: Number of physical qubits and reaction limit of all Grover's searches in `NVSieve` and `GaussSieve` with and without LSH/LSF as a function of the lattice dimension D. We assume an underlying active-volume physical architecture. The reaction limits also include the classical hashing times. The quantities are computed based on heuristic assumptions described in the main text.

is noticeable the decrease in resources as more sophisticated hashing techniques are employed, from angular LSH to spherical LSH and LSF. Spherical LSH is more expensive than angular LSH in lower dimensions due to high lower-order terms, specially coming from the $O(2^{\sqrt{D}})$ hashing time. It is, however, asymptotically better than angular LSH as expected. At the range of proposed cryptographic dimensions $D \approx 400$, the best attack (`GaussSieve` with spherical LSF) requires around $\approx 10^{13}$ physical qubits and $\approx 10^{31}$ years to find a lattice's shortest vector. We note that most crossovers between classical and quantum time complexities happen after dimension 200, or dimension 300 for `GaussSieve` specifically.

In Appendix A we revisit the heuristic assumptions made in this section and compare the performance of all Grover's searches under these assumptions to the performance using data from numerical simulations on classical hardware. In other words, we perform resource estimates using the evolution of the list L in a real run of `GaussSieve` on classical hardware. As a brief summary, the time complexities reported in this section can probably be reduced by half under more realistic and thorough heuristic assumptions.



(a) Physical qubits of `GaussSieve` without QRAM



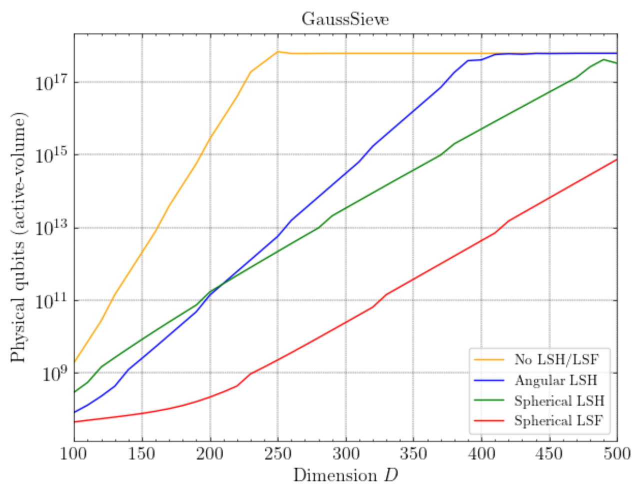
(b) Reaction limit of `GaussSieve` without QRAM

Figure 6: Number of physical qubits and reaction limit of `GaussSieve` with and without LSH/LSF as a function of the lattice dimension D in the scenario where QRAM has negligible cost. We assume an underlying active-volume physical architecture. The quantities are computed based on heuristic assumptions described in the main text.

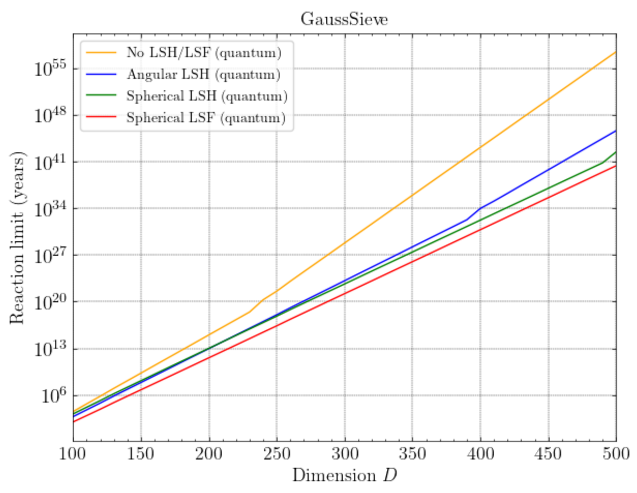
8.3 The cost of QRAM

From the previous sections, specially from the cost expressions of Section 8.1, it should be clear that QRAM is the most expensive component in our quantum circuits. The need to access an exponentially large dataset imposes a huge burden on size. Not only that, but the loss of sequential access to the dataset set by Grover's search implies that, when using any hashing technique, we must first gather all candidate vectors and store them separately in order to later use QRAM. For these reasons, we analyse in this section the required resources for sieving algorithms under the scenario where QRAM has negligible cost, akin to Albrecht et al. [14]. This is done by repeating the procedure from the previous sections, but this time zeroing out all contributions from QRAM to Toffoli-count, logical qubits, active volume, and reaction depth in the expressions from Section 8.1. For simplicity, we focus on `GaussSieve`, as it requires less resources than `NVSieve` and performs classically better in practice. The number of physical qubits under an active-volume architecture and reaction limit for `GaussSieve` with and without QRAM are compared in Figure 6.

The absence of QRAM has little impact on the reaction limit of `GaussSieve` for dimensions up to 500. According to Sections 6 and 8.1, QRAM is a shallow circuit with reaction depth of $2\lceil\log_2|L|\rceil - 2 = 196$ for $|L| = 2^{0.193\cdot 500+2.325} \approx 5.61\cdot 10^{29}$, while the arithmetic part of one Grover iteration has reaction depth of $2(1+\lceil\log_2 D\rceil)(\kappa-1) + 2\kappa\log_2\kappa - 2\kappa - 2\log_2\kappa + 4 = 870$, hence why there is no noticiable change in the reaction limit from Figure 6b. On the other hand, however, the number of physical qubits is drastically reduced from $\approx 10^{25}$ down to $\approx 4\cdot 10^8$ for `GaussSieve` with spherical LSF at dimension $D = 500$, for example. Such drastic change is expected, since a bucket-brigade-style QRAM with shallow reaction depth requires a number of logical qubits roughly equal to the size of the list stored in memory. We note that earlier resources estimates on Shor's algorithm placed the number of physical qubits to be in the range 10^8-10^{10} [193, 110, 79, 163, 84], which is comparable to our estimates of running `GaussSieve` without QRAM in high dimensions. From Figure 6a it can be noted that the number of physical qubits has little dependence on the employed hashing techniques. Without QRAM, the number of physical qubits comes mainly from the arithmetic modules, which are independent of the list size. Finally, the sudden changes in the number of physical qubits from Figure 6a are due to integer increases in the code distance d in order to maintain the error rates below 0.1%.



(a) Physical qubits of `GaussSieve` with limited depth



(b) Reaction limit of `GaussSieve` with limited depth

Figure 7: Number of physical qubits and reaction limit of `GaussSieve` with and without LSH/LSF as a function of the lattice dimension D in the scenario where the reaction depth of each Grover's seach is at most 2^{40}. We assume an underlying active-volume physical architecture. The quantities are computed based on heuristic assumptions described in the main text.

8.4 Depth constraints: NIST standardisation

In many realistic situations, a quantum attacker would have bounded resources, e.g., be constrained by a total running time or circuit depth. In its call for proposals for the post-quantum cryptography standardisation process [162], NIST introduced the parameter `MAXDEPTH` to bound the circuit depth of any potential attacker, suggesting reasonably values in the range of 2^{40} to 2^{96} logical gates. As explained in their proposal [162, Section 4.A.5], the value 2^{40} is "the approximate number of gates that presently envisioned quantum computing architectures are expected to serially perform in a year" [110], while 2^{96} is "the approximate number of gates that atomic scale qubits with speed of light propagation times could perform in a millennium". In this section, we revisit the results from Section 8.2 and constrain the circuit depth. Since several quantities could be interpreted as the circuit depth, we set the parameter `MAXDEPTH` as a limit to the reaction depth of any Grover's search. This means that, for `MAXDEPTH` $= 2^{40}$, any Grover's search would be time limited to 2^{40} μs ≈ 12.73 days, assuming a reaction time of 1 μs, while for `MAXDEPTH` $= 2^{96}$, any Grover's search would be time limited to 2^{96} μs $\approx 2.51 \cdot 10^{15}$ years. This, in turns, limits the number of Grover iterations. In order to meet the maximum reaction depth, we split the list L in `GaussSieve` (list of centers S in `NVSieve` and list of candidates C when using LSH/LSF) into F disjoint parts, each to be searched by a different instance of Grover algorithm. The number F of sequential Grover's searches required to set a maximum reaction depth of I in `GaussSieve` is thus determined by the equation

$$\begin{aligned} I = \lceil 9.2 \log_3(1/\delta)\sqrt{|L|/F}\rceil \big(&2\lceil\log_2(|L|/F)\rceil + 2(1+\lceil\log_2 D\rceil)(\kappa-1) \\ &+ 2\kappa\log_2\kappa - 2\kappa - 2\log_2\kappa + 2 + 2\lceil\log_2\lceil\log_2(|L|/F)\rceil\rceil\big), \end{aligned}$$

which simply follows from the reaction-depth expression from Section 8.1.2. A similar equation to determining F holds for `NVSieve`. Here $2^{40} \leq I \leq 2^{96}$ as set by NIST. The value F obtained from the above equation is then used to determine other quantities like number of physical qubits.

In Figure 7 we depict the number of physical qubits and total reaction limit of `GaussSieve` with and without LSH/LSF in the scenario where each Grover's search has reaction depth at most 2^{40}. As usual, the total number of physical qubits is the maximum number of physical qubits used by any Grover's search, while the total reaction limit is the sum of the reaction limits of all Grover's searches. For small dimensions, the reaction limit of Grover's search is smaller than `MAXDEPTH` $= 2^{40}$, so there are no differences between Figures 5 and 7. However, the depth restriction begins to take effect for dimensions higher than ≈ 250. As a consequence, the number of physical qubits becomes mostly constant since only lists of at most a certain size can be searched. On the other hand, the reaction limit of the whole algorithm increases more rapidly with the dimension D, since employing F sequential Grover's searches over list of size $|L|/F$ is less time efficient than employing a single Grover's search over the whole list L. The end result is a considerable decrease in number of physical qubits, while the time increases by a few orders of magnitude, specially in `GaussSieve` without LSH/LSF, whose circuit depth is capped in smaller dimensions. A similar effect would be observed for a different `MAXDEPTH`. We remark that the reaction depth of Grover's search is smaller than 2^{96} for all dimensions $D \leq 500$, hence why we omit an analysis for $\texttt{MAXDEPTH} = 2^{96}$.

9 Discussions and open problems

In this paper, we considered the most important sieving algorithms (`NVSieve` and `GaussSieve`) and gave rigorous estimates on the time and space required to execute internal searching subroutines with Grover's search. Our estimation analysis took into consideration fixed-point quantum arithmetic, the cost of QRAM, different physical architectures like baseline and active-volume one, and quantum error correction. For the sake of comparison, we also consider equivalent classical implementations where Grover's search was replaced with sequential classical searching operations. We note that using BKZ to break the security of level-1 NIST candidate cryptosystems like Kyber-512, Falcon-512, and DiLithium require us to solve SVP in dimensions (block sizes of) over 400. At this lattice dimension, even `GaussSieve` with spherical LSF under an active-volume architecture would require $\approx 10^{13}$ physical qubits and $\approx 10^{31}$ years to execute all Grover's search subroutines, which also takes into consideration classical hashing operations but ignores memory allocation. Most of the required qubits are due to QRAM, meaning that any quantum advantage will only be possible if QRAM becomes substantially less costly. On the other hand, a single-core classical computer with 6 GHs clock rate would also require $\approx 10^{31}$ years to solve SVP at dimension 400, meaning that there is little advantage at dimensions of cryptographic interest.

We have not explored the possibility of parallelising the list search by breaking it into smaller parts and employing different Grover's searches on each part. However, it is well known that Grover's search does not parallelise well [203], meaning that F parallel Grover algorithms running on F separate search spaces have a total width that is larger by a factor of F compared to a single Grover algorithm on the whole search space while only reducing the depth by a factor of $\sqrt{F}$. We expect to observe a decrease in total runtime (Figure 5) via parallelisation by n order of magnitude in exchange to an increase in number of physical qubits by roughly $2n$ orders of magnitude.

The hash parameter k and the number of hash tables t were chosen so that nearby vectors collide with high probability and the classical time hashing is balanced out by the quantum time searching. A very precise choice for t would require sorting out the constant factors in each of these complexities, which we did not consider. We leave it to future works a more careful analysis on the choice of k, t, α.

We saw that the introduction of LSH or LSF requires a classical pre-search to gather all candidate vectors from the buckets with the same hash as a given vector and place them on a QRAM. Albrecht et al. [14] (partially)[2] evaded such a problem by employing just one hash table and considering more than one bucket via a "XOR and population count trick". By using their popcount filter and amplifying the amplitude of the vectors that pass such filter via quantum amplitude amplification [46], Grover's search is performed only on a subset of vectors which are close to the target vector with high probability. This lessens the cost coming from quantum arithmetic circuits in Grover's oracle. Even though Albrecht et al. obtained a cost expression for this "filtered" Grover's search, it requires strong bounds on the number of solutions M. In particular, the authors assumed that the number of solutions to the filtered search is known exactly beforehand, which we deem too strong of an assumption. It would be interesting to obtain more rigorous cost expressions on their filtered Grover's search (akin to Ref. [50]) and perform a complete resource estimation on sieving algorithms employing it.

[2]The problem is still present when considering LSF in their `ListDecodingSearch` [14, Algorithm 4].

Finally, we leave to future works a rigorous resource estimate on the quantum-random-walk-based sieving algorithm of Chailloux and Loyer [54] and on enumeration algorithms and the consideration of metrics other than time and number of physical qubits like energy consumption.

Acknowledgements

We thank Divesh Aggrawal, Martin Albrecht, Hugo Cable, Anupam Chattopadhyay, Craig Gidney, András Gilyén, Daniel Litinski, Markus Müller, and Adithya Sireesh for useful discussions. JFD acknowledges funding from ERC grant No. 810115-DYNASNET. Research at CQT is funded by the National Research Foundation, the Prime Minister's Office, and the Ministry of Education, Singapore under the Research Centres of Excellence programme's research grant R-710-000-012-135. We also acknowledge funding from the Quantum Engineering Programme (QEP 2.0) under grant NRF2021-QEP2-02-P05. This work was done in part while JFD was visiting the Simons Institute for the Theory of Computing, supported by NSF QLCI Grant No. 2016245.

References

[1] https://github.com/TheCharmingSociopath/qsieve. 8

[2] Dimitris Achlioptas. Database-friendly random projections. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, page 274–281, New York, NY, USA, 2001. Association for Computing Machinery. 28

[3] Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003. Special Issue on PODS 2001. 28

[4] Divesh Aggarwal, Yanlin Chen, Rajendra Kumar, and Yixin Shen. Improved classical and quantum algorithms for the shortest vector problem via bounded distance decoding. *arXiv preprint arXiv:2002.07955*, 2020. 4

[5] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete Gaussian sampling: Extended abstract. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 733–742, New York, NY, USA, 2015. Association for Computing Machinery. 3, 4, 23

[6] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8):2201–2214, 2002. 3, 23

[7] D. Aharonov and M. Ben-Or. Fault-tolerant quantum computation with constant error. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 176–188, New York, NY, USA, 1997. Association for Computing Machinery. 14

[8] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 99–108, New York, NY, USA, 1996. Association for Computing Machinery. 2

[9] Miklós Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 10–19, New York, NY, USA, 1998. Association for Computing Machinery. 23

[10] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 284–293, New York, NY, USA, 1997. Association for Computing Machinery. 23

[11] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. An overview of the sieve algorithm for the shortest lattice vector problem. In Joseph H. Silverman, editor, *Cryptography and Lattices*, pages 1–3, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. 3, 23, 24

[12] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, STOC '01, page 601–610, New York, NY, USA, 2001. Association for Computing Machinery. 3, 23, 24

[13] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 717–746, Cham, 2019. Springer International Publishing. 4

[14] Martin R. Albrecht, Vlad Gheorghiu, Eamonn W. Postlethwaite, and John M. Schanck. Estimating quantum speedups for lattice sieves. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 583–613, Cham, 2020. Springer International Publishing. 3, 4, 6, 7, 44, 46

[15] Martin R. Albrecht, Miloš Prokop, Yixin Shen, and Petros Wallden. Variational quantum solutions to the Shortest Vector Problem. *Quantum*, 7:933, March 2023. 4

[16] Jonathan Allcock, Jinge Bao, João F Doriguello, Alessandro Luongo, and Miklos Santha. Constant-depth circuits for Uniformly Controlled Gates and Boolean functions with application to quantum memory circuits. *arXiv preprint arXiv:2308.08539*, 2023. 21

[17] Mishal Almazrooie, Azman Samsudin, Rosni Abdullah, and Kussay N. Mutter. Quantum reversible circuit of AES-128. *Quantum Information Processing*, 17(5):112, Mar 2018. 4

[18] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John Schanck. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In Roberto Avanzi and Howard Heys, editors, *Selected Areas in Cryptography – SAC 2016*, pages 317–337, Cham, 2017. Springer International Publishing. 4

[19] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830, 2013. 17

[20] Alexandr Andoni, Piotr Indyk, Huy L. Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, page 1018–1028, USA, 2014. Society for Industrial and Applied Mathematics. 3, 11, 12

[21] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 793–801, New York, NY, USA, 2015. Association for Computing Machinery. 3, 11, 12

[22] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. *arXiv preprint arXiv:1501.01062*, 2015. 12

[23] Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. On the robustness of bucket brigade quantum RAM. *New Journal of Physics*, 17(12):123010, 2015. 5, 22

[24] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, Oct 2019. 5

[25] Ryan Babbush, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. Encoding electronic spectra in quantum circuits with linear T complexity. *Phys. Rev. X*, 8:041015, Oct 2018. 5, 23

[26] Hafiz Md. Hasan Babu. Cost-efficient design of a quantum multiplier–accumulator unit. *Quantum Information Processing*, 16(1):30, Dec 2016. 18

[27] Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: Algorithm specifications and supporting documentation (version 3.1), 2021. 4, 35

[28] Shi Bai, Maya-Iggy van Hoof, Floyd B. Johnson, Tanja Lange, and Tran Ngo. Concrete analysis of quantum lattice enumeration. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 131–166, Singapore, 2023. Springer Nature Singapore. 3, 4

[29] C. J. Ballance, T. P. Harty, N. M. Linke, M. A. Sepiol, and D. M. Lucas. High-fidelity quantum logic gates using trapped-ion hyperfine qubits. *Phys. Rev. Lett.*, 117:060504, Aug 2016. 13

[30] F Battistel, C Chamberland, K Johar, R W J Overwater, F Sebastiano, L Skoric, Y Ueno, and M Usman. Real-time decoding for fault-tolerant quantum computing: progress, challenges and outlook. *Nano Futures*, 7(3):032003, aug 2023. 14

[31] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, jul 2001. 20

[32] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, page 10–24, USA, 2016. Society for Industrial and Applied Mathematics. 3, 4, 10, 12, 13, 29, 30, 35

[33] Anja Becker, Nicolas Gama, and Antoine Joux. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. *Cryptology ePrint Archive*, 2015. 3, 4

[34] Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope LSH. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2016*, pages 3–23, Cham, 2016. Springer International Publishing. 3

[35] David Beckman, Amalavoyal N. Chari, Srikrishna Devabhaktuni, and John Preskill. Efficient networks for quantum factoring. *Phys. Rev. A*, 54:1034–1063, Aug 1996. 17

[36] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997. 20

[37] Charles H. Bennett, David P. DiVincenzo, John A. Smolin, and William K. Wootters. Mixed-state entanglement and quantum error correction. *Phys. Rev. A*, 54:3824–3851, Nov 1996. 13

[38] Daniel J. Bernstein and Tanja Lange. Post-quantum cryptography. *Nature*, 549(7671):188–194, Sep 2017. 3

[39] Nina Bindel, Xavier Bonnetain, Marcel Tiepelt, and Fernando Virdia. Quantum lattice enumeration in limited depth. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, pages 72–106, Cham, 2024. Springer Nature Switzerland. 3, 4

[40] H. Bombin. Topological order with a twist: Ising anyons from an Abelian model. *Phys. Rev. Lett.*, 105:030403, Jul 2010. 14

[41] Hector Bombin, Isaac H Kim, Daniel Litinski, Naomi Nickerson, Mihir Pant, Fernando Pastawski, Sam Roberts, and Terry Rudolph. Interleaving: Modular architectures for fault-tolerant photonic quantum computing. *arXiv preprint arXiv:2103.08612*, 2021. 5, 14

[42] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. Quantum Security Analysis of AES. *IACR Transactions on Symmetric Cryptology*, 2019(2):55–93, June 2019. 4

[43] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998. 5, 20

[44] P.O. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan. On universal and fault-tolerant quantum computing: a novel basis and a new constructive proof of universality for Shor's basis. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 486–494, 1999. 9

[45] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3), July 2014. 3

[46] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002. 4, 20, 21, 46

[47] Sergey Bravyi and Jeongwan Haah. Magic-state distillation with low overhead. *Phys. Rev. A*, 86:052329, Nov 2012. 16

[48] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Phys. Rev. A*, 71:022316, Feb 2005. 5, 15, 16

[49] Benjamin J. Brown, Katharina Laubscher, Markus S. Kesselring, and James R. Wootton. Poking holes and cutting corners to achieve Clifford gates with the surface code. *Phys. Rev. X*, 7:021029, May 2017. 14

[50] Chris Cade, Marten Folkertsma, Ido Niesen, and Jordi Weggemans. Quantifying Grover speed-ups beyond asymptotic analysis. *Quantum*, 7:1133, October 2023. 5, 21, 46

[51] A. R. Calderbank and Peter W. Shor. Good quantum error-correcting codes exist. *Phys. Rev. A*, 54:1098–1105, Aug 1996. 13

[52] Earl T. Campbell and Mark Howard. Unified framework for magic state distillation and multi-qubit gate synthesis with reduced resource cost. *Phys. Rev. A*, 95:022316, Feb 2017. 16

[53] Earl T. Campbell and Mark Howard. Magic state parity-checker with pre-distilled components. *Quantum*, 2:56, March 2018. 16

[54] André Chailloux and Johanna Loyer. Lattice sieving via quantum random walks. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 63–91, Cham, 2021. Springer International Publishing. 4, 47

[55] Christopher Chamberland and Earl T. Campbell. Universal quantum computing with twist-free and temporally encoded lattice surgery. *PRX Quantum*, 3:010331, Feb 2022. 5, 14

[56] Christopher Chamberland, Kyungjoo Noh, Patricio Arrangoiz-Arriola, Earl T. Campbell, Connor T. Hann, Joseph Iverson, Harald Putterman, Thomas C. Bohdanowicz, Steven T. Flammia, Andrew Keller, Gil Refael, John Preskill, Liang Jiang, Amir H. Safavi-Naeini, Oskar Painter, and Fernando G.S.L. Brandão. Building a fault-tolerant quantum computer using concatenated cat codes. *PRX Quantum*, 3:010329, Feb 2022. 5, 14

[57] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, page 380–388, New York, NY, USA, 2002. Association for Computing Machinery. 3, 4, 11

[58] Zijun Chen, Kevin J. Satzinger, Juan Atalaya, Alexander N. Korotkov, Andrew Dunsworth, Daniel Sank, Chris Quintana, Matt McEwen, Rami Barends, Paul V. Klimov, Sabrina Hong, Cody Jones, Andre Petukhov, Dvir Kafri, Sean Demura, Brian Burkett, Craig Gidney, Austin G. Fowler, Alexandru Paler, Harald Putterman, Igor Aleiner, Frank Arute, Kunal Arya, Ryan Babbush, Joseph C. Bardin, Andreas Bengtsson, Alexandre Bourassa, Michael Broughton, Bob B. Buckley, David A. Buell, Nicholas Bushnell, Benjamin Chiaro, Roberto Collins, William Courtney, Alan R. Derk, Daniel Eppens, Catherine Erickson, Edward Farhi, Brooks Foxen, Marissa Giustina, Ami Greene, Jonathan A. Gross, Matthew P. Harrigan, Sean D. Harrington, Jeremy Hilton, Alan Ho, Trent Huang, William J. Huggins, L. B. Ioffe, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Kostyantyn Kechedzhi, Seon Kim, Alexei Kitaev, Fedor Kostritsa, David Landhuis, Pavel Laptev, Erik Lucero, Orion Martin, Jarrod R. McClean, Trevor McCourt, Xiao Mi, Kevin C. Miao, Masoud Mohseni, Shirin Montazeri, Wojciech Mruczkiewicz, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Michael Newman, Murphy Yuezhen Niu, Thomas E. O'Brien, Alex Opremcak, Eric Ostby, Bálint Pató, Nicholas Redd, Pedram Roushan, Nicholas C. Rubin, Vladimir Shvarts, Doug Strain, Marco Szalay, Matthew D. Trevithick, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Juhwan Yoo, Adam Zalcman, Hartmut Neven, Sergio Boixo, Vadim Smelyanskiy, Yu Chen, Anthony Megrant, Julian Kelly, and Google Quantum AI. Exponential suppression of bit or phase errors with cyclic error correction. *Nature*, 595(7867):383–387, Jul 2021. 14

[59] Craig R. Clark, Holly N. Tinkey, Brian C. Sawyer, Adam M. Meier, Karl A. Burkhardt, Christopher M. Seck, Christopher M. Shappert, Nicholas D. Guise, Curtis E. Volin, Spencer D. Fallek, Harley T. Hayden, Wade G. Rellergert, and Kenton R. Brown. High-fidelity Bell-state preparation with $^{40}\mathrm{Ca}^{+}$ optical qubits. *Phys. Rev. Lett.*, 127:130505, Sep 2021. 13

[60] Andrew N. Cleland. An introduction to the surface code. *SciPost Phys. Lect. Notes*, page 49, 2022. 14

[61] John Horton Conway and Neil James Alexander Sloane. *Sphere packings, lattices and groups*, volume 290. Springer Science & Business Media, 2013. 31

[62] Steven A Cuccaro, Thomas G Draper, Samuel A Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit. *arXiv preprint quant-ph/0410184*, 2004. 17

[63] Nicolas Delfosse and Naomi H. Nickerson. Almost-linear time decoding algorithm for topological codes. *Quantum*, 5:595, December 2021. 5, 15

[64] Nicolas Delfosse and Gilles Zémor. Linear-time maximum likelihood decoding of surface codes over the quantum erasure channel. *Phys. Rev. Res.*, 2:033042, Jul 2020. 5, 15

[65] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 09 2002. 5, 14, 15

[66] Olivia Di Matteo, Vlad Gheorghiu, and Michele Mosca. Fault-tolerant resource estimation of quantum random-access memories. *IEEE Transactions on Quantum Engineering*, 1:1–13, 2020. 5, 23

[67] David P. DiVincenzo. Two-bit gates are universal for quantum computation. *Phys. Rev. A*, 51:1015–1022, Feb 1995. 9

[68] João F. Doriguello, Alessandro Luongo, Jinge Bao, Patrick Rebentrost, and Miklos Santha. Quantum algorithm for stochastic optimal stopping problems with applications in finance. In François Le Gall and Tomoyuki Morimae, editors, *17th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2022)*, volume 232 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:24, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 17

[69] T.G. Draper, S.A. Kutin, E.M. Rains, and K.M. Svore. A logarithmic-depth quantum carry-lookahead adder. *Quantum Information and Computation*, 6(4&5):351–369, 07 2006. 17

[70] Thomas G Draper. Addition on a quantum computer. *arXiv preprint quant-ph/0008033*, 2000. 17

[71] Guillaume Duclos-Cianci and David Poulin. Reducing the quantum-computing overhead with complex gate distillation. *Phys. Rev. A*, 91:042315, Apr 2015. 16

[72] Guillaume Duclos-Cianci and Krysta M. Svore. Distillation of nonstabilizer states for universal quantum computation. *Phys. Rev. A*, 88:042325, Oct 2013. 16

[73] Bryan Eastin and Emanuel Knill. Restrictions on transversal encoded quantum gate sets. *Phys. Rev. Lett.*, 102:110502, Mar 2009. 15

[74] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. 5, 15

[75] Irene Gil Fernández, Jaehoon Kim, Hong Liu, and Oleg Pikhurko. New lower bounds on kissing numbers and spherical codes in high dimensions. *arXiv preprint arXiv:2111.01255*, 2021. 31

[76] U. Fincke and M. Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44(170):463–471, 1985. 3, 23

[77] Austin G. Fowler, Simon J. Devitt, and Cody Jones. Surface code implementation of block code state distillation. *Scientific Reports*, 3(1):1939, Jun 2013. 16

[78] Austin G Fowler and Craig Gidney. Low overhead quantum computation using lattice surgery. *arXiv preprint arXiv:1808.06709*, 2018. 5, 14, 15, 16

[79] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86:032324, Sep 2012. 14, 44

[80] J. P. Gaebler, T. R. Tan, Y. Lin, Y. Wan, R. Bowler, A. C. Keith, S. Glancy, K. Coakley, E. Knill, D. Leibfried, and D. J. Wineland. High-fidelity universal gate set for $^9\mathrm{Be}^+$ ion qubits. *Phys. Rev. Lett.*, 117:060505, Aug 2016. 13

[81] S. S. Gayathri, R. Kumar, Samiappan Dhanalakshmi, Brajesh Kumar Kaushik, and Majid Haghparast. T-count optimized wallace tree integer multiplier for quantum computing. *International Journal of Theoretical Physics*, 60(8):2823–2835, Aug 2021. 18

[82] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery. 3

[83] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, page 197–206, New York, NY, USA, 2008. Association for Computing Machinery. 24

[84] Vlad Gheorghiu and Michele Mosca. Benchmarking the quantum cryptanalysis of symmetric, public-key and hash-based cryptographic schemes. *arXiv preprint arXiv:1902.02332*, 2019. 44

[85] Craig Gidney. Halving the cost of quantum addition. *Quantum*, 2:74, June 2018. 5, 17

[86] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, April 2021. 6, 15

[87] Craig Gidney and Austin G. Fowler. Efficient magic state factories with a catalyzed $|CCZ\rangle$ to $2|T\rangle$ transformation. *Quantum*, 3:135, April 2019. 5, 16

[88] Craig Gidney, Noah Shutty, and Cody Jones. Magic state cultivation: growing T states as cheap as CNOT gates. *arXiv preprint arXiv:2409.17595*, 2024. 7

[89] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Architectures for a quantum random access memory. *Physical Review A*, 78(5):052310, 2008. 3, 5, 22

[90] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Physical review letters*, 100(16):160501, 2008. 3, 5, 22

[91] Phil Gossett. Quantum carry-save arithmetic. *arXiv preprint quant-ph/9808061*, 1998. 17

[92] Daniel Gottesman. Class of quantum error-correcting codes saturating the quantum Hamming bound. *Phys. Rev. A*, 54:1862–1868, Sep 1996. 13

[93] Daniel Gottesman. *Stabilizer codes and quantum error correction.* PhD thesis, California Institute of Technology, United States – California, 1997. 13, 14

[94] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying Grover's algorithm to AES: Quantum resource estimates. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography*, pages 29–43, Cham, 2016. Springer International Publishing. 4

[95] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery. 3, 20

[96] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79:325–328, Jul 1997. 3, 20

[97] Jeongwan Haah and Matthew B. Hastings. Codes and Protocols for Distilling T, controlled-S, and Toffoli Gates. *Quantum*, 2:71, June 2018. 5, 16

[98] Connor T. Hann. *Practicality of Quantum Random Access Memory.* PhD thesis, Yale University, 2021. 21

[99] Connor T. Hann, Gideon Lee, S.M. Girvin, and Liang Jiang. Resilience of quantum random access memory to generic noise. *PRX Quantum*, 2:020311, Apr 2021. 5, 22

[100] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In Yeow Meng Chee, Zhenbo Guo, San Ling, Fengjing Shao, Yuansheng Tang, Huaxiong Wang, and Chaoping Xing, editors, *Coding and Cryptology*, pages 159–190, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 24

[101] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe P. Buhler, editor, *Algorithmic Number Theory*, pages 267–288, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. 2, 3

[102] Dominic Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, dec 2012. 5, 14, 15

[103] Security Innovation Inc. NTRU challenge parameter sets and public keys. https://web.archive.org/web/20160310141551/https://www.securityinnovation.com/uploads/ntru-challenge-parameter-sets-and-public-keys-new.pdf. 3

[104] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 604–613, New York, NY, USA, 1998. Association for Computing Machinery. 3, 10, 11

[105] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing Grover oracles for quantum key search on AES and LowMC. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 280–310, Cham, 2020. Springer International Publishing. 4

[106] Samuel Jaques and Arthur G. Rattew. QRAM: A survey and critique. *arXiv preprint arXiv:2305.10310*, 2023. 21

[107] H. V. Jayashree, Himanshu Thapliyal, Hamid R. Arabnia, and V. K. Agrawal. Ancilla-input and garbage-output optimized design of a reversible quantum integer multiplier. *The Journal of Supercomputing*, 72(4):1477–1493, Apr 2016. 17, 18

[108] Cody Jones. Low-overhead constructions for the fault-tolerant Toffoli gate. *Phys. Rev. A*, 87:022328, Feb 2013. 17

[109] Cody Jones. Multilevel distillation of magic states for quantum computing. *Phys. Rev. A*, 87:042305, Apr 2013. 16

[110] N. Cody Jones, Rodney Van Meter, Austin G. Fowler, Peter L. McMahon, Jungsang Kim, Thaddeus D. Ladd, and Yoshihisa Yamamoto. Layered architecture for quantum computing. *Phys. Rev. X*, 2:031007, Jul 2012. 44, 45

[111] G. A. Kabatjanskiĭ and V. I. Levenšteĭn. Bounds for packings on the sphere and in space. *Problemy Peredači Informacii*, 14(1):3–25, 1978. 30

[112] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, page 193–206, New York, NY, USA, 1983. Association for Computing Machinery. 3, 23

[113] David Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. *J. ACM*, 45(2):246–265, mar 1998. 12

[114] Subhash Khot. Hardness of approximating the shortest vector problem in lattices. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '04, page 126–135, USA, 2004. IEEE Computer Society. 23

[115] Subhash Khot. Hardness of approximating the shortest vector problem in lattices. *J. ACM*, 52(5):789–808, sep 2005. 23

[116] Hyunji Kim, Kyoungbae Jang, Yujin Oh, Woojin Seok, Wonhuck Lee, Kwangil Bae, Ilkwon Sohn, and Hwajeong Seo. Finding shortest vector using quantum NV sieve on Grover. In Hwajeong Seo and Suhri Kim, editors, *Information Security and Cryptology – ICISC 2023*, pages 97–118, Singapore, 2024. Springer Nature Singapore. 3

[117] Hyunji Kim, Kyungbae Jang, Hyunjun Kim, Anubhab Baksi, Sumanta Chakraborty, and Hwajeong Seo. Quantum NV sieve on Grover for solving shortest vector problem. *Cryptology ePrint Archive*, 2024. 3, 4

[118] Elena Kirshanova, Erik Mårtensson, Eamonn W. Postlethwaite, and Subhayan Roy Moulik. Quantum algorithms for the approximate k-list problem and their application to lattice sieving. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 521–551, Cham, 2019. Springer International Publishing. 4

[119] Elena Kirshanova, Alexander May, and Julian Nowakowski. New NTRU records with improved lattice bases. In Thomas Johansson and Daniel Smith-Tone, editors, *Post-Quantum Cryptography*, pages 167–195, Cham, 2023. Springer Nature Switzerland. 3

[120] A Yu Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191, dec 1997. 5, 14

[121] A.Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003. 5, 14

[122] Philip Klein. Finding the closest lattice vector when it's unusually close. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, page 937–941, USA, 2000. Society for Industrial and Applied Mathematics. 24

[123] Emanuel Knill and Raymond Laflamme. Concatenated quantum codes. *arXiv preprint quant-ph/9608012*, 1996. 14

[124] Emanuel Knill, Raymond Laflamme, and Wojciech H. Zurek. Resilient quantum computation. *Science*, 279(5349):342–345, 1998. 14

[125] Saurabh Kotiyal, Himanshu Thapliyal, and Nagarajan 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*, pages 545–550, 2014. 18

[126] Sebastian Krinner, Nathan Lacroix, Ants Remm, Agustin Di Paolo, Elie Genois, Catherine Leroux, Christoph Hellings, Stefania Lazar, Francois Swiadek, Johannes Herrmann, Graham J. Norris, Christian Kraglund Andersen, Markus Müller, Alexandre Blais, Christopher Eichler, and Andreas Wallraff. Realizing repeated quantum error correction in a distance-three surface code. *Nature*, 605(7911):669–674, May 2022. 14

[127] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 3–22, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. 3, 4, 11, 25, 28, 34

[128] Thijs Laarhoven. *Search problems in cryptography: from fingerprinting to lattice sieving*. PhD thesis, Mathematics and Computer Science, February 2016. Proefschrift. 3, 4, 8, 11, 12, 24, 25, 28, 29, 30, 34, 35

[129] Thijs Laarhoven and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In Kristin Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology – LATINCRYPT 2015*, pages 101–118, Cham, 2015. Springer International Publishing. 3, 4, 11, 29

[130] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77(2):375–400, Dec 2015. 3, 28, 29, 34

[131] Raymond Laflamme, Cesar Miquel, Juan Pablo Paz, and Wojciech Hubert Zurek. Perfect quantum error correcting code. *Phys. Rev. Lett.*, 77:198–201, Jul 1996. 13

[132] Yongjae Lee and Woo Chang Kim. Concise formulas for the surface area of the intersection of two hyperspherical caps. Technical report, KAIST, 2014. 8

[133] Hai-Sheng Li, Ping Fan, Haiying Xia, and Gui-Lu Long. The circuit design and optimization of quantum multiplier and divider. *Science China Physics, Mechanics & Astronomy*, 65(6):260311, Apr 2022. 17, 18

[134] Hai-Sheng Li, Ping Fan, Haiying Xia, Huiling Peng, and Gui-Lu Long. Efficient quantum arithmetic operation circuits for quantum image processing. *Science China Physics, Mechanics & Astronomy*, 63(8):280311, Jun 2020. 17, 18

[135] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 287–296, New York, NY, USA, 2006. Association for Computing Machinery. 28

[136] Shengqiao Li. Concise formulas for the area and volume of a hyperspherical cap. *Asian Journal of Mathematics & Statistics*, 4(1):66–70, 2010. 8

[137] Chia-Chun Lin, Amlan Chakrabarti, and Niraj K. Jha. Qlib: Quantum module library. *J. Emerg. Technol. Comput. Syst.*, 11(1), oct 2014. 17, 18

[138] Daniel Litinski. A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery. *Quantum*, 3:128, March 2019. 5, 14, 15, 16

[139] Daniel Litinski. Magic State Distillation: Not as Costly as You Think. *Quantum*, 3:205, December 2019. 5, 15, 16

[140] Daniel Litinski. How to compute a 256-bit elliptic curve private key with only 50 million Toffoli gates. *arXiv preprint arXiv:2306.08585*, 2023. 6, 15, 23

[141] Daniel Litinski. Quantum schoolbook multiplication with fewer Toffoli gates. *arXiv preprint arXiv:2410.00899*, 2024. 5, 18

[142] Daniel Litinski and Naomi Nickerson. Active volume: An architecture for efficient fault-tolerant quantum computers with limited non-local connections. *arXiv preprint arXiv:2211.15465*, 2022. 5, 14, 15, 16, 17, 18, 23

[143] Daniel Litinski and Felix von Oppen. Lattice surgery with a twist: simplifying Clifford gates of surface codes. *Quantum*, 2:62, May 2018. 14, 15

[144] Daniel Litinski and Felix von Oppen. Quantum computing with Majorana fermion codes. *Phys. Rev. B*, 97:205404, May 2018. 15

[145] Guang Hao Low, Vadym Kliuchnikov, and Luke Schaeffer. Trading T gates for dirty qubits in state preparation and unitary synthesis. *Quantum*, 8:1375, June 2024. 23

[146] Ivaylo S. Madjarov, Jacob P. Covey, Adam L. Shaw, Joonhee Choi, Anant Kale, Alexandre Cooper, Hannes Pichler, Vladimir Schkolnik, Jason R. Williams, and Manuel Endres. High-fidelity entanglement and detection of alkaline-earth Rydberg atoms. *Nature Physics*, 16(8):857–861, Aug 2020. 13

[147] Frederic Magniez, Ashwin Nayak, Jeremie Roland, and Miklos Santha. Search via quantum walk. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, page 575–584, New York, NY, USA, 2007. Association for Computing Machinery. 4

[148] Kooroush Manochehri, Mehrshad Khosraviani, and Sina Mirshafiee. A regular architecture for a low-quantum-cost n-bit multiplier. *Computers and Electrical Engineering*, 114:109061, 2024. 18

[149] Artur Mariano, Özgür Dagdelen, and Christian Bischof. A comprehensive empirical comparison of parallel ListSieve and GaussSieve. In Luís Lopes, Julius Žilinskas, Alexandru Costan, Roberto G. Cascella, Gabor Kecskemeti, Emmanuel Jeannot, Mario Cannataro, Laura Ricci, Siegfried Benkner, Salvador Petit, Vittorio Scarano, José Gracia, Sascha Hunold, Stephen L. Scott, Stefan Lankes, Christian Lengauer, Jesus Carretero, Jens Breitbart, and Michael Alexander, editors, *Euro-Par 2014: Parallel Processing Workshops*, pages 48–59, Cham, 2014. Springer International Publishing. 6, 31, 32

[150] Adam M. Meier, Bryan Eastin, and Emanuel Knill. Magic-state distillation with the four-qubit code. *Quantum Info. Comput.*, 13(3–4):195–209, mar 2013. 16

[151] Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, page 92, USA, 1998. IEEE Computer Society. 23

[152] Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, 2001. 23

[153] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 700–718, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 2

[154] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, pages 147–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. 23

[155] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, page 351–358, New York, NY, USA, 2010. Association for Computing Machinery. 3, 23

[156] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, page 1468–1480, USA, 2010. Society for Industrial and Applied Mathematics. 3, 4, 23, 29, 30, 31, 32

[157] Ilya N. Moskalenko, Ilya A. Simakov, Nikolay N. Abramov, Alexander A. Grigorev, Dmitry O. Moskalev, Anastasiya A. Pishchimova, Nikita S. Smirnov, Evgeniy V. Zikiy, Ilya A. Rodionov, and Ilya S. Besedin. High fidelity two-qubit gates on fluxoniums using a tunable coupler. *npj Quantum Information*, 8(1):130, Nov 2022. 13

[158] Priyanka Mukhopadhyay. A quantum random access memory (QRAM) using a polynomial encoding of binary strings. *arXiv preprint arXiv:2408.16794*, 2024. 7

[159] Edgard Muñoz-Coreas and Himanshu Thapliyal. Quantum circuit design of a T-count optimized integer multiplier. *IEEE Transactions on Computers*, 68(5):729–739, 2019. 17, 18

[160] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008. 3, 4, 6, 24, 25, 27

[161] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010. 9, 14, 15

[162] NIST. Post-quantum cryptography: Call for proposals. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Call-for-Proposals. Accessed: 2024-02-05. 3, 8, 45

[163] Joe O'Gorman and Earl T. Campbell. Quantum computation with realistic magic-state factories. *Phys. Rev. A*, 95:032338, Mar 2017. 16, 44

[164] Takuya Ohno, Gaku Arakawa, Ikuo Ichinose, and Tetsuo Matsui. Phase structure of the random-plaquette Z_2 gauge model: accuracy threshold for a toric quantum memory. *Nuclear Physics B*, 697(3):462–480, 2004. 14

[165] F. Orts, E. Filatovas, G. Ortega, J. F. SanJuan-Estrada, and E. M. Garzón. Improving the number of T gates and their spread in integer multipliers on quantum computing. *Phys. Rev. A*, 107:042621, Apr 2023. 18

[166] F. Orts, G. Ortega, E.F. Combarro, and E.M. Garzón. A review on reversible quantum adders. *Journal of Network and Computer Applications*, 170:102810, 2020. 17

[167] Koustubh Phalak, Avimita Chatterjee, and Swaroop Ghosh. Quantum random access memory for dummies. *arXiv preprint arXiv:2305.01178*, 2023. 21

[168] Michael Pohst. On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. *SIGSAM Bull.*, 15(1):37–44, feb 1981. 3, 23

[169] Ehsan PourAliAkbar and Mohammad Mosleh. An efficient design for reversible Wallace unsigned multiplier. *Theoretical Computer Science*, 773:43–52, 2019. 18

[170] John Preskill. Reliable quantum computers. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):385–410, 1998. 14

[171] Milos Prokop, Petros Wallden, and David Joseph. Grover's oracle for the shortest vector problem and its application in hybrid classical-quantum solvers. *arXiv preprint arXiv:2402.13895*, 2024. 3, 4

[172] Xavier Pujol and Damien Stehle. Solving the shortest lattice vector problem in time $2^{2.465n}$. Cryptology ePrint Archive, Paper 2009/605, 2009. 24

[173] Oded Regev. New lattice-based cryptographic constructions. *J. ACM*, 51(6):899–942, nov 2004. 23

[174] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA, 2005. Association for Computing Machinery. 2

[175] Oded Regev. Lattice-based cryptography. In *Proceedings of the 26th Annual International Conference on Advances in Cryptology*, CRYPTO'06, page 131–141, Berlin, Heidelberg, 2006. Springer-Verlag. 2, 23

[176] Ben W. Reichardt. Quantum universality from magic states distillation applied to CSS codes. *Quantum Information Processing*, 4(3):251–264, Aug 2005. 5, 16

[177] Lidia Ruiz-Perez and Juan Carlos Garcia-Escartin. Quantum arithmetic with the quantum Fourier transform. *Quantum Information Processing*, 16(6):152, Apr 2017. 18

[178] Andrei Ruskuc, Chun-Ju Wu, Jake Rochman, Joonhee Choi, and Andrei Faraon. Nuclear spin-wave quantum register for a solid-state qubit. *Nature*, 602(7897):408–413, Feb 2022. 13

[179] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. P. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown, T. M. Gatterman, S. K. Halit, K. Gilmore, J. A. Gerber, B. Neyenhuis, D. Hayes, and R. P. Stutz. Realization of real-time fault-tolerant quantum error correction. *Phys. Rev. X*, 11:041058, Dec 2021. 14

[180] Yuval R. Sanders, Dominic W. Berry, Pedro C.S. Costa, Louis W. Tessler, Nathan Wiebe, Craig Gidney, Hartmut Neven, and Ryan Babbush. Compilation of fault-tolerant quantum heuristics for combinatorial optimization. *PRX Quantum*, 1:020312, Nov 2020. 18

[181] Michael Schneider. Analysis of gauss-sieve for solving the shortest vector problem in lattices. In *Proceedings of the 5th International Conference on WALCOM: Algorithms and Computation*, WALCOM'11, page 89–97, Berlin, Heidelberg, 2011. Springer-Verlag. 31, 32

[182] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In L. Budach, editor, *Fundamentals of Computation Theory*, pages 68–85, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg. 23

[183] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(1):181–199, Aug 1994. 3, 23

[184] Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52:R2493–R2496, Oct 1995. 13

[185] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999. 2

[186] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. 2

[187] P.W. Shor. Fault-tolerant quantum computation. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 56–65, 1996. 9

[188] Thomas M. Stace and Sean D. Barrett. Error correction and degeneracy in surface codes suffering loss. *Phys. Rev. A*, 81:022317, Feb 2010. 14

[189] A. M. Steane. Error correcting codes in quantum theory. *Phys. Rev. Lett.*, 77:793–797, Jul 1996. 13

[190] Andrew Steane. Multiple-particle interference and quantum error correction. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 452(1954):2551–2577, 1996. 13

[191] Ashley M. Stephens. Fault-tolerant thresholds for quantum error correction with the surface code. *Phys. Rev. A*, 89:022321, Feb 2014. 14

[192] Zedong Sun, Chunxiang Gu, and Yonghui Zheng. A review of sieve algorithms in solving the shortest lattice vector problem. *IEEE Access*, 8:190475–190486, 2020. 3

[193] Rodney Van Meter, Thaddeus D Ladd, Austin G Fowler, and Yoshihisa Yamamoto. Distributed quantum computation architecture using semiconductor nanophotonics. *International Journal of Quantum Information*, 8(01n02):295–323, 2010. 44

[194] Vlatko Vedral, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Phys. Rev. A*, 54:147–153, Jul 1996. 17

[195] Chenyang Wang, Jim Harrington, and John Preskill. Confinement-Higgs transition in a disordered gauge theory and the accuracy threshold for quantum memory. *Annals of Physics*, 303(1):31–58, 2003. 14

[196] David S. Wang, Austin G. Fowler, and Lloyd C. L. Hollenberg. Surface code quantum computing with error rates over 1%. *Phys. Rev. A*, 83:020302, Feb 2011. 14

[197] Xiaoyun Wang, Mingjie Liu, Chengliang Tian, and Jingguo Bi. Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, page 1–9, New York, NY, USA, 2011. Association for Computing Machinery. 3

[198] Adam Wills, Min-Hsiu Hsieh, and Hayata Yamasaki. Constant-overhead magic state distillation. *arXiv preprint arXiv:2408.07764*, 2024. 7

[199] Ronald de Wolf. Quantum computing: Lecture notes. *arXiv preprint arXiv:1907.09415*, 2019. 9, 21

[200] Siyi Yang, Naixu Guo, Miklos Santha, and Patrick Rebentrost. Quantum Alphatron: quantum advantage for learning with kernels and noise. *Quantum*, 7:1174, November 2023. 17

[201] Christof Zalka. Fast versions of Shor's quantum factoring algorithm. *arXiv preprint quant-ph/9806084*, 1998. 17

[202] Christof Zalka. A Grover-based quantum search of optimal order for an unknown number of marked elements. *arXiv preprint quant-ph/9902049*, 1999. 20

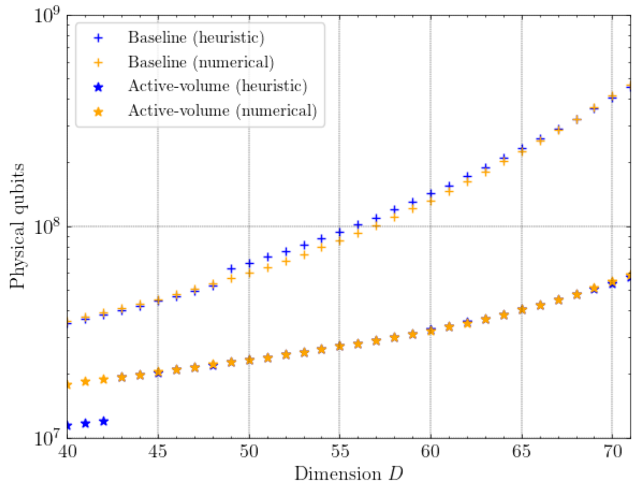
[203] Christof Zalka. Grover's quantum searching algorithm is optimal. *Phys. Rev. A*, 60:2746–2751, Oct 1999. 7, 20, 46

[204] Feng Zhang, Yanbin Pan, and Gengran Hu. A three-level sieve algorithm for the shortest vector problem. In Tanja Lange, Kristin Lauter, and Petr Lisoněk, editors, *Selected Areas in Cryptography – SAC 2013*, pages 29–47, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. 3

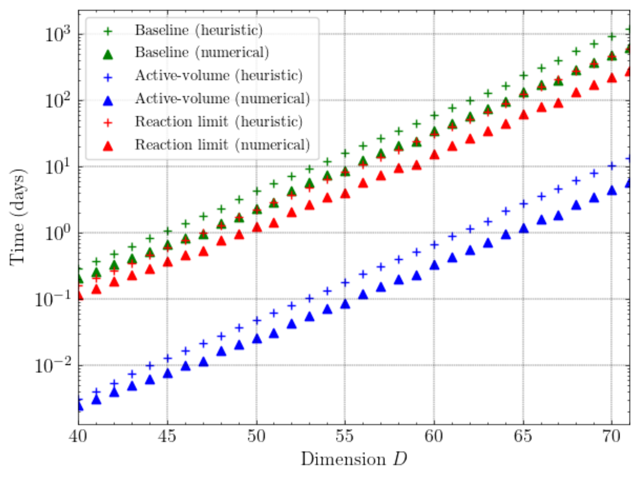
A Comparison between heuristic assumptions and numerical simulations

Some of the heuristic assumptions from Section 8.2 are worst-case simplifications, e.g., the assumption that any Grover's search in `NVSieve` and `GaussSieve` has at most one solution, or that the list size is maximum throughout all iterations. In reality, we expect `NVSieve` and `GaussSieve` to perform better than described in Section 8.2: the list size should be quite smaller in many iterations than its maximum size at the end of the algorithm, and several solutions could exist when performing Grover's search.

In this appendix, we compare the results of Section 8.2 to those obtained from actual numerical simulations. To be more precise, we solved SVP on a random lattice of dimension $40 \leq D \leq 71$ using `GaussSieve` (without LSH/LSF) on a classical hardware and recorded the list L_i and number of solutions M_i at each step i. This creates a history of list and number-of-solution pairs $\{(L_i, M_i)\}_i$. Given a list size $|L_i|$ and a number of solutions M_i, it is then possible to estimate the amount of



(a) Physical qubits of `GaussSieve`



(b) Time estimate of `GaussSieve`

Figure 8: Comparison between quantum resources of `GaussSieve` obtained through heuristic assumptions and through numerical data. (a) Comparison between physical qubits under baseline and active-volume architectures. (b) Comparison between reaction limit and circuit time under baseline and active-volume architectures. The heuristic data is obtained through the assumptions from Section 8.2, while the numerical data is obtained by running `GaussSieve` on a classical hardware and employing its internal parameters (list size and number of solutions) at every step.

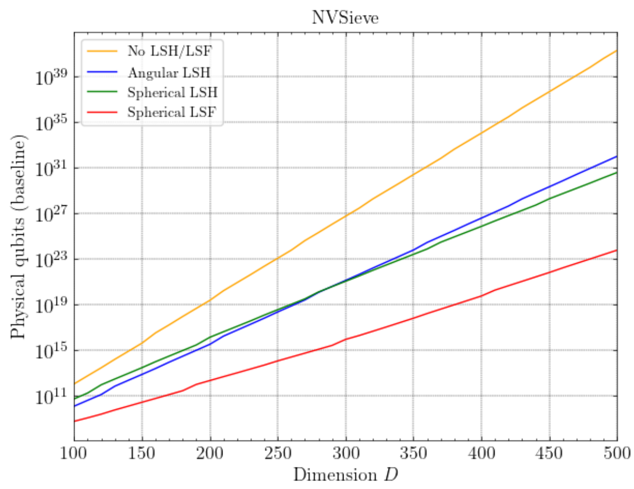
resources that would be required by Grover's search at that given step i of `GaussSieve` by following Section 8.1. The total amount of physical qubits employed by one particular `GaussSieve` run is the maximum number of physical qubits that would be required for any search step i, while the total quantum time complexity due to all Grover algorithms is the sum of the quantum time complexities of each individual search step i. Since `GaussSieve` is a randomised algorithm, we repeat this procedure a few times and take averages of the final number of physical qubits and quantum time complexity[3]. The results are displayed in Figure 8.

Figure 8a compares the number of physical qubits, both under baseline and active-volume architectures, that result from following the heuristic assumptions of Section 8.2 and from considering the history of list and number of solutions $\{(L_i, M_i)\}_i$ of an average run of `GaussSieve`. There is little difference between both approaches in the number of physical qubits, which is to be expected since the number of physical qubits is a function of the maximum list size and its heuristic value of $2^{0.193D+2.325}$ is a fitting of actual numerical data. More interestingly, though, Figure 8b compares several time complexities (reaction limit and circuit time under baseline and active-volume architectures) between heuristic and numerical data. As anticipated, we can observe that `GaussSieve` has lower quantum time complexities in "practice" than under the heuristic assumptions of Section 8.2. The improvement in time complexity is around 50%, meaning that `GaussSieve` with Grover's search should be two times faster than reported in Section 8 for dimensions $40 \leq D \leq 71$. It is not unreasonable to extend such advantage to larger dimensions and to `GaussSieve` with hashing techniques.

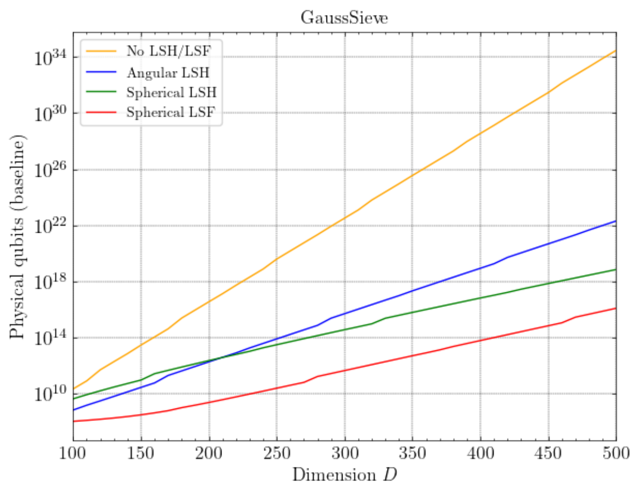
B Extra results

In this appendix we provide more results that were omitted from Section 8, e.g., number of physical qubits and circuit time under baseline and active-volume physical architectures for both `NVSieve` and `GaussSieve` with and without LSH/LSF. Figures 9 to 11 describe results for `NVSieve` and `GaussSieve` with QRAM, without QRAM, and with Grover's searches reaction-depth limited to 2^{40}, respectively.

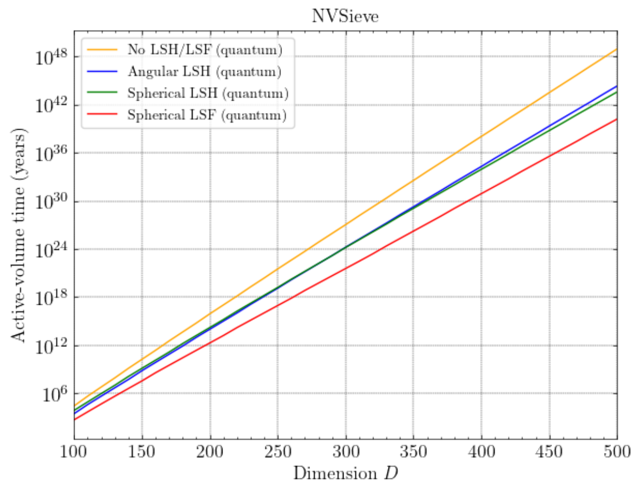
[3]For $40 \leq D \leq 70$ we repeated the procedure 10 times, while for $D = 71$ we repeated it 3 times.



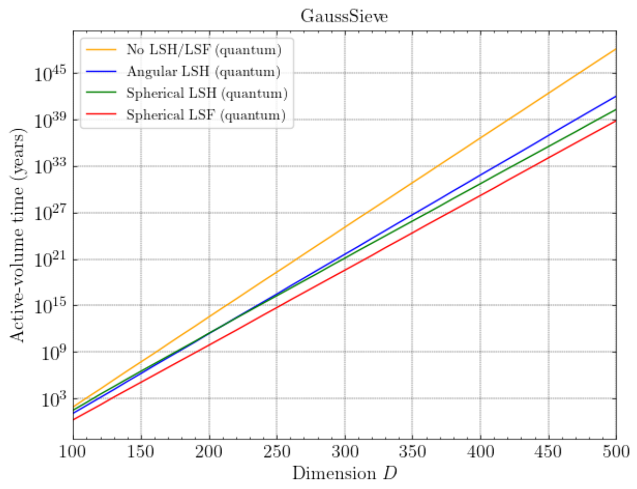
(a) Baseline physical qubits of `NVSieve`



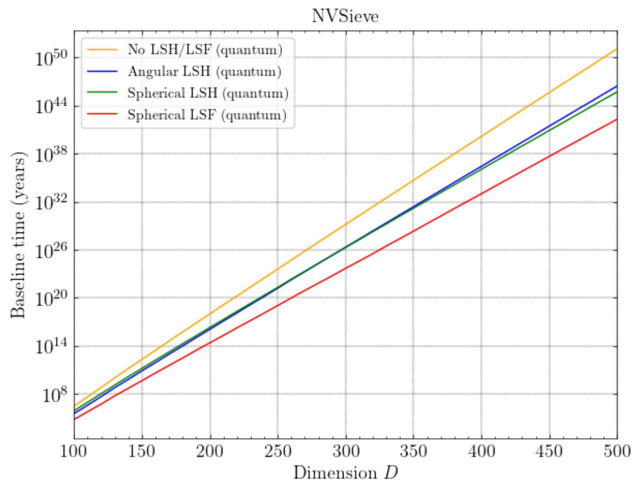
(b) Baseline physical qubits of `GaussSieve`



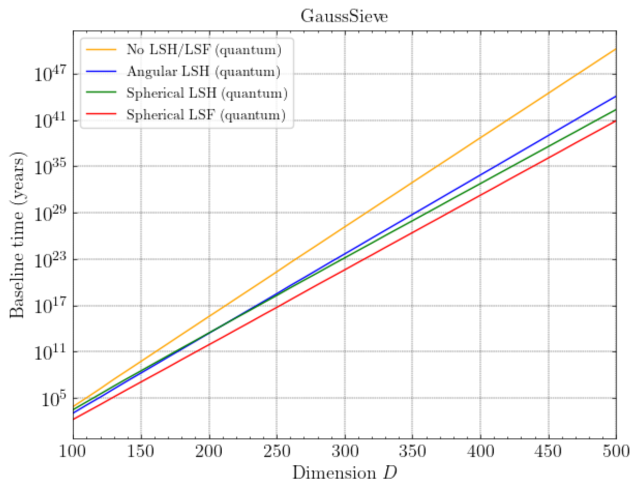
(c) Active-volume circuit time of `NVSieve`



(d) Active-volume circuit time of `GaussSieve`

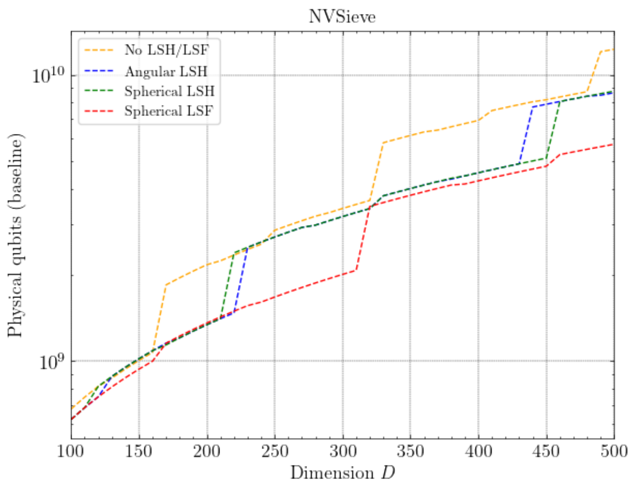


(e) Baseline circuit time of `NVSieve`

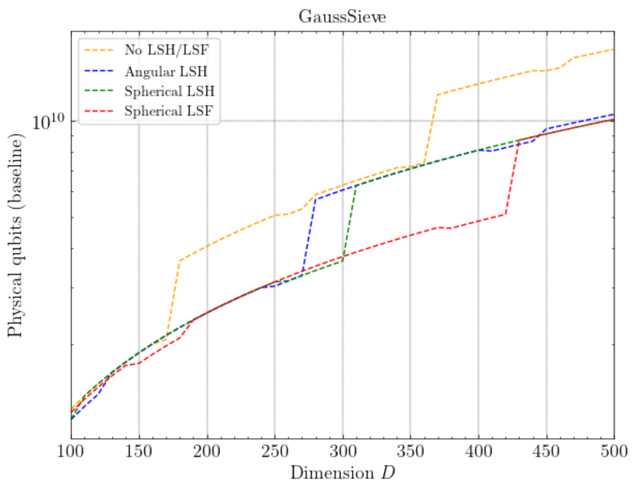


(f) Baseline circuit time of `GaussSieve`

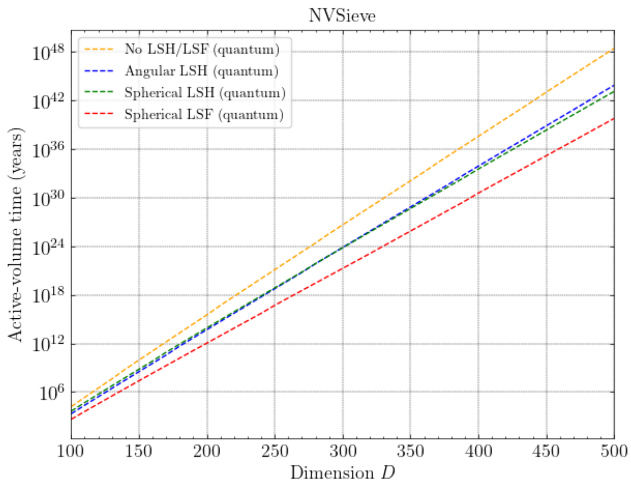
Figure 9: Number of physical qubits and circuit times in `NVSieve` and `GaussSieve` with and without LSH/LSF under baseline and active-volume physical architectures as a function of lattice dimension D. Reaction limits and circuit time also include classical hashing time.



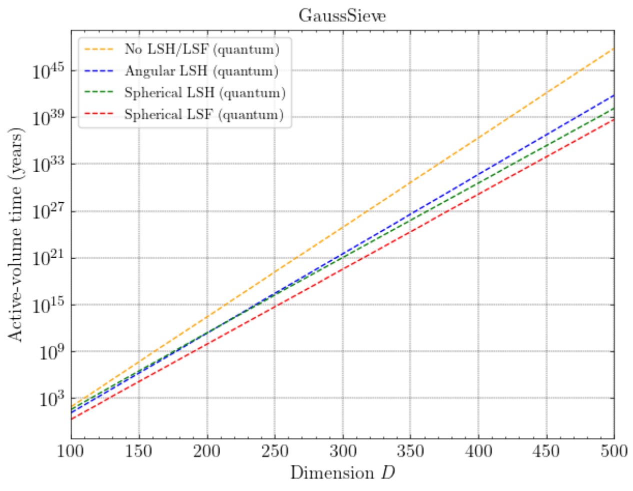
(a) Baseline physical qubits of `NVSieve` without QRAM



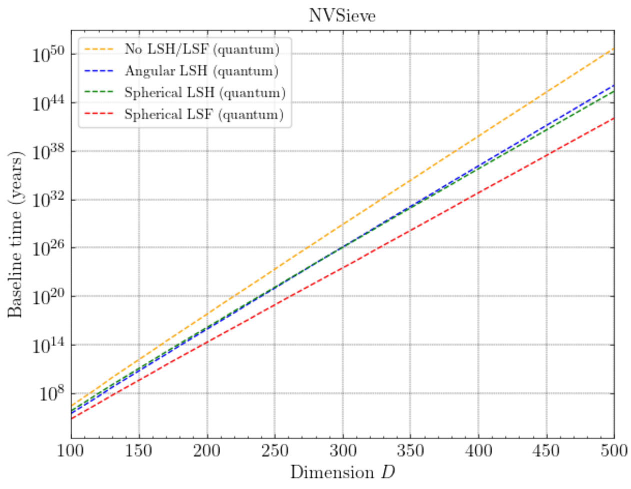
(b) Baseline physical qubits of `GaussSieve` without QRAM



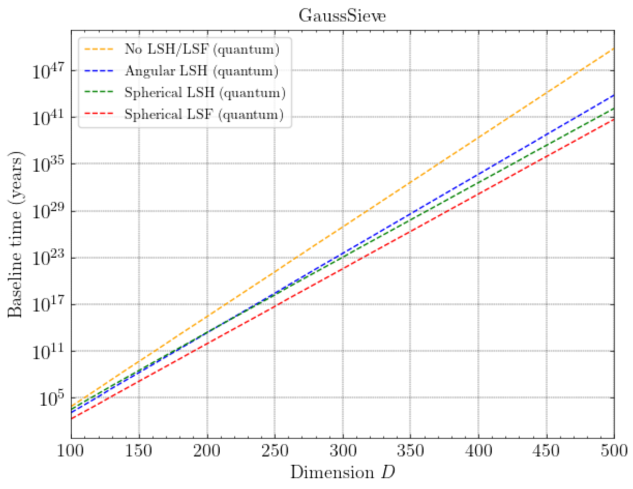
(c) Active-volume circuit time of `NVSieve` without QRAM



(d) Active-volume circuit time of `GaussSieve` without QRAM

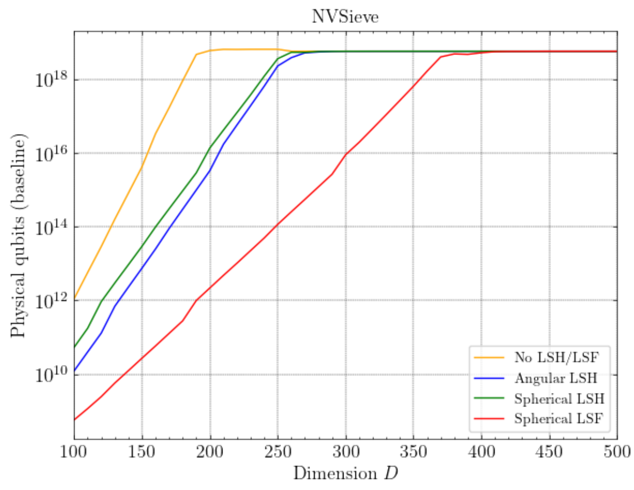


(e) Baseline circuit time of `NVSieve` without QRAM

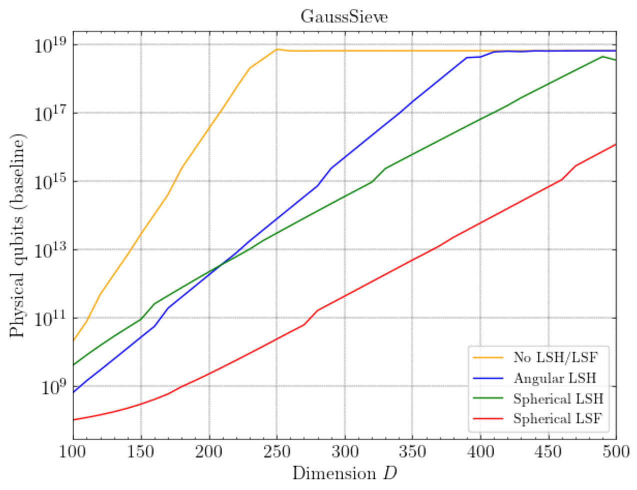


(f) Baseline circuit time of `GaussSieve` without QRAM

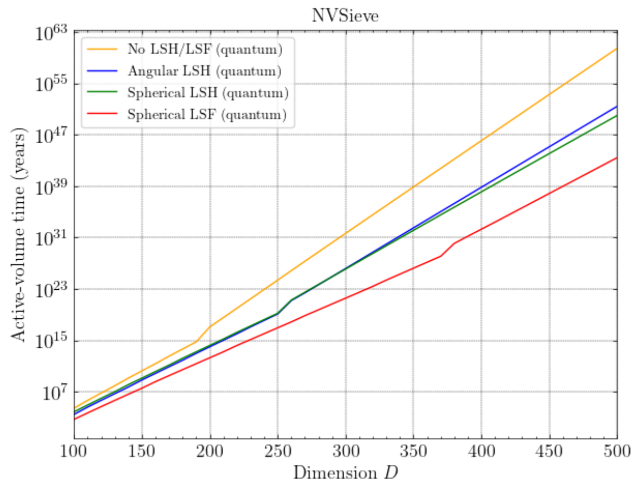
Figure 10: Number of physical qubits and circuit times in `NVSieve` and `GaussSieve` with and without LSH/LSF under baseline and active-volume physical architectures as a function of lattice dimension D in the scenario where QRAM has negligible cost. Reaction limits and circuit time also include classical hashing time.



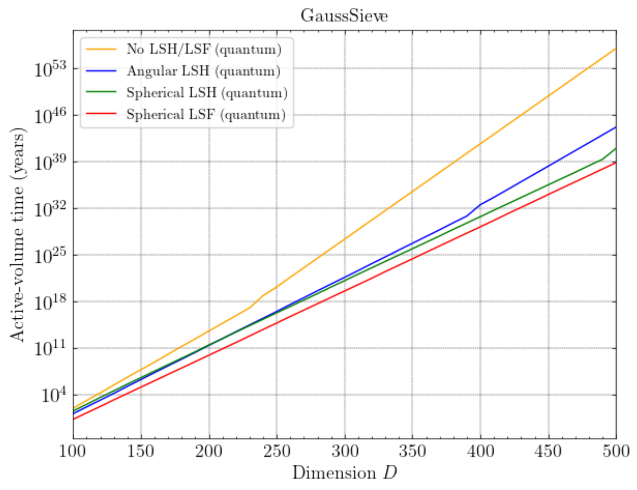
(a) Baseline physical qubits of `NVSieve` with limited depth



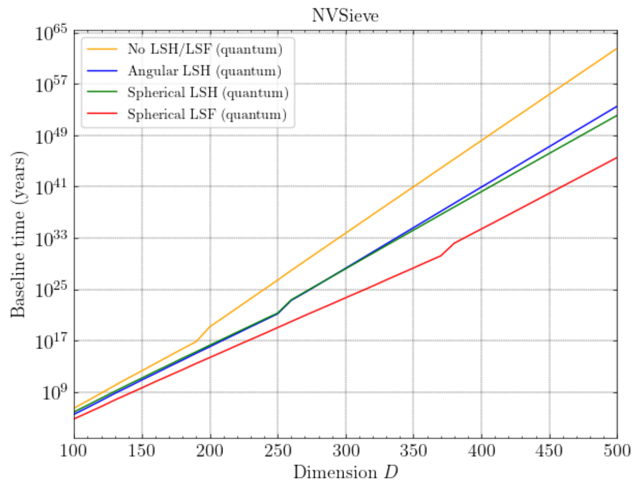
(b) Baseline physical qubits of `GaussSieve` with limited depth



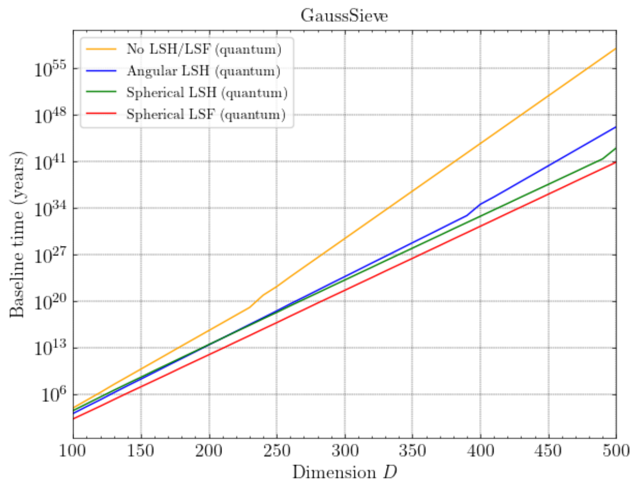
(c) Active-volume circuit time of `NVSieve` with limited depth



(d) Active-volume circuit time of `GaussSieve` with limited depth



(e) Baseline circuit time of `NVSieve` with limited depth



(f) Baseline circuit time of `GaussSieve` with limited depth

Figure 11: Number of physical qubits and circuit times in `NVSieve` and `GaussSieve` under baseline and active-volume physical architectures as a function of lattice dimension D in the scenario where the reaction depth of each Grover's search is at most 2^{40}. Circuit time includes classical hashing time.