Does quantum lattice sieving require quantum RAM?

Beomgeun Cho[1], Minki Hhan*[2], Taehyun Kim[1], Jeonghoon Lee[1], and Yixin Shen[3]

[1] Seoul National University, Seoul, Republic of Korea, {c11sh0117,taehyun,dalsan2113}@snu.ac.kr
[2] The University of Texas at Austin, Texas, USA, minki.hhan@austin.utexas.edu
[3] Univ Rennes, Inria, CNRS, IRISA, Rennes, France yixin.shen@inria.fr

Abstract. In this paper, we study the requirement for *quantum* random access memory (QRAM) in quantum lattice sieving, a fundamental algorithm for lattice-based cryptanalysis.
First, we obtain a lower bound on the cost of quantum lattice sieving with a bounded size QRAM. We do so in a new query model encompassing a wide range of lattice sieving algorithms similar to those in the classical sieving lower bound by Kirshanova and Laarhoven [CRYPTO 21]. This implies that, under reasonable assumptions, quantum speedups in lattice sieving require the use of QRAM. In particular, no quantum speedup is possible without QRAM.
Second, we investigate the trade-off between the size of QRAM and the quantum speedup. We obtain a new interpolation between classical and quantum lattice sieving. Moreover, we show that further improvements require a novel way to use the QRAM by proving the optimality of some subroutines. An important caveat is that this trade-off requires a strong assumption on the efficient replacement of QRAM data, indicating that even speedups with a small QRAM are already challenging.
Finally, we provide a circuit for quantum lattice sieving without using QRAM. Our circuit has a better depth complexity than the best classical algorithms but requires an exponential amount of qubits. To the best of our knowledge, this is the first quantum speedup for lattice sieving without QRAM in the standard quantum circuit model. We explain why this circuit does not contradict our lower bound, which considers the query complexity.

Keywords: lattice sieving, the shortest vector problem, collision finding, quantum RAM

1 Introduction

One of the major impacts of quantum computing is to efficiently solve the integer factoring problem with Shor's algorithm [57], threatening the currently

* Most of this work was done while Minki Hhan was in KIAS, Korea.

used cryptographic schemes like the RSA cryptosystem [55]. As a countermeasure, NIST has started to standardize post-quantum cryptography (PQC) to replace the currently deployed ones. Lattice-based cryptography is one of the most promising post-quantum cryptography candidates due to its provable security [54] and efficiency. Several schemes [9,20,27] have thus been selected to be standardized by NIST [51].

The post-quantum security of lattice-based cryptography has been studied extensively, e.g. in [7,6,58,48]. Many lattice cryptanalysis algorithms heavily rely on lattice reduction algorithms [36,3,6,21,26,31,34,44,53], which in turn reduce to solving the shortest vector problem (SVP); therefore the SVP algorithm often dominates the cost of the overall attack. Classically, lattice sieving and enumeration are the most promising approaches to solving the SVP. For the SVP over a d-dimensional lattice, lattice sieving has time complexity $2^{O(d)}$ but requires an exponential memory $2^{O(d)}$ as well [2,49]. On the other hand, enumeration algorithms only require a polynomial-size memory while having super-exponential complexity $2^{O(d \log d)}$ [52].

A series of works have shown how to obtain asymptotic quantum speedups for both lattice algorithms. For enumeration, asymptotic quadratic quantum speedup [8] have been obtained using the quantum backtracking technique [47], positively answering the conjectured quadratic speedup in [37]. The complexity of quantum lattice sieving [42,33,18,14] is more involved. Compared to the best classical time complexity $2^{0.2925d+o(d)}$ [12], the current best quantum time complexity is $2^{0.2563d+o(d)}$ which is achieved using (reusable) quantum walk techniques [14]. The concrete complexity of the quantum enumeration and sieving has been explored in [10] and [5] respectively, which demonstrate that quantum speedups for lattice cryptanalysis are achievable with full-fledged quantum computers.

Our current understanding of quantum computer architectures, however, has raised many concerns about full-fledged, unbounded-depth, error-free quantum computers. Two notable constraints on quantum devices must be considered—the *quantum circuit depth* and the *quantum random access memory (QRAM)*. Since qubits suffer from physical errors and decoherence, achieving large quantum circuit depth could be challenging. The limited quantum circuit depth model has thus been suggested by NIST [1, Section 4.A.5] in their standardization procedure.

QRAM is a quantum variant of classical RAM, allowing coherent access operation

$$|i\rangle |0\rangle \mapsto |i\rangle |x_i\rangle$$

given a stored list $L = (x_i)_{i \in I}$ of classical data. This operation requires coherent access to all data in L and thus relies on a new quantum architecture. Many architectures [29,28,46] have been suggested. However, all known proposals suffer from some drawbacks. [38] discusses some fundamental limitations on these infrastructures. Based on the conclusion of [38], QRAM should be considered expensive or potentially unrealistic. Ideally, we would like to remove the use of QRAM in quantum algorithms to avoid the above issues. This is, for example,

the case of the quantum collision finding problem where a quantum speedup without QRAM was achieved in [19].

The quantum circuit depths and/or QRAM may be a hurdle for the quantum lattice algorithms. The quantum sieving estimation [5] shows that all known quantum sieving algorithms require huge quantum depth and QRAM, leaving the question of whether large QRAM and/or large quantum circuit depth are fundamental barriers to achieving a quantum advantage.

This question is also relevant for enumeration where a recent work [13] explored quantum speedups in the bounded quantum circuit depth model and concluded that the current quantum enumeration techniques are unlikely to provide practical speedup in this setting.

In this paper, we mainly focus on the QRAM aspect of the question:

Does quantum lattice sieving necessarily require a large QRAM?

Our main question is particularly interesting in the near future where small-sized quantum computers are available, but QRAM does not exist. Furthermore, if the answer is yes, and it turns out that efficient QRAM is unlikely to exist and large quantum circuit depth is hard to realize, then the implication combining [13] could be even more striking—*there could be no quantum speedup in lattice cryptanalysis!*

We would like to stress that quantum lattice sieving algorithms have been continuously improving since [5], and the QRAM-less variant has not been explored yet. This is in contrast with the classical setting where much more is known. Kirshanova and Laarhoven [40] proved the tightness of the best classical lattice sieving [12] in the nearest-neighbor model that encompasses most sieving algorithms. To the best of our knowledge, no similar results exist in the quantum setting.[1]

1.1 This work

In this paper, we study the role of QRAM[2] in quantum lattice sieving. We provide some evidence that QRAM access is essential for quantum speedups and explore the fine-grained trade-off between the size of QRAM and the time complexity of quantum lattice sieving.

Need for (large) QRAM in quantum lattice sieving. We introduce a new quantum and classical lattice sieving (or near-neighbor) model that encompasses all known sieving algorithms [18,14,33,42]. This model is similar to the one used in the classical sieving lower bound [40]. We then show that, in this model, any quantum algorithm beyond the classical lower bound must use a QRAM. Putting

[1] [40] claims that the quantum algorithm of [42], which uses Grover's algorithm, is optimal. However, [18], which uses quantum walks, is already better than [42].

[2] This paper focuses on the QRAM structure accessing classical data, usually referred to as QRACM. We refer to Section 2.3 or [38] for more discussion on QRAM for quantum data.

it differently, our lower bound suggests that the QRAM-less quantum speedup for lattice sieving requires a fundamentally new idea.

In fact, we prove a more general lower bound, which implies that a stronger quantum speedup in lattice sieving requires a larger QRAM. While our trade-off bound is not tight, it asserts the need for a somewhat large QRAM to achieve the complexity as in the current best quantum lattice sieving algorithm.

We note that our model includes two types of strategies to include the two algorithms using quantum walks [18,14], which slightly deviate from the framework of [40][3]. For the lower bound regarding the strategy including [18,14], we make some reasonable assumptions about the use of QRAM: Roughly, it says that the efficient generation of coherent states of the vectors can only be achieved through the QRAM access.

Quantum lattice sieving with small QRAM. Given the requirement of QRAM in the quantum speedup, we explore the fine-grained trade-off between the size of QRAM and the quantum complexity of lattice sieving. We revisit the trade-off algorithms that smoothly interpolate between the known classical and quantum complexity of lattice sieving, which is also briefly discussed in [18].

The graph showing the overall trade-off can be found in Figure 2. While the trade-off is mostly unchanged from [18] in the (very) small QRAM regime, we found that the bounded QRAM version of [33] gives a better time complexity than the one suggested in [18] for a moderate size QRAM. These trade-offs are obtained using variants of Grover's algorithm with a bounded QRAM. We show that further improvements require a new way to use the QRAM by proving the optimality of some subroutines.[4]

Another important caveat we found is that these trade-off algorithms require a strong assumption on the QRAM: The stored classical data must be replaceable very efficiently, i.e. in $2^{o(s)}$ time for a QRAM of size 2^s. This efficient operation potentially forces a very specific implementation of QRAM or asks for multiple QRAMs to virtually implement such an operation. We refer to Section 4.4 for a more detailed discussion. We are not aware of any way to use small QRAM in lattice sieving without this assumption. This indicates that the quantum speedup for lattice sieving is challenging even with a moderate size QRAM.

These findings prompt us to revisit the comparison between enumeration and sieving for the famous lattice reduction algorithm BKZ [56]. If we assume that arbitrary depth quantum circuits are available but we cannot build large QRAM, then quantum enumeration will outperform quantum sieving even for relatively large dimension. Figure 1 plots the complexity of BKZ using quantum enumeration [4, Fig. 10] (assuming a full quadratic speedup of [8]) compared to our lower bound on sieving with no QRAM of Section 6. We also include the best quantum algorithm for BKZ with sieving, assuming no constraint on the QRAM size [14]. Note that all algorithms above achieve the same root Hermite factor $k^{1/k}$, and that we have neglected some polynomial factors. The goal of

[3] Though their classical lower bound still applies (as we showed).
[4] See Theorem 3. We are not aware of a similar lower bound in the literature.

this comparison is therefore not to give exact values for the cross-over points between enumeration and sieving but rather to give orders of magnitude and to observe the impact of the (lack of) QRAM.

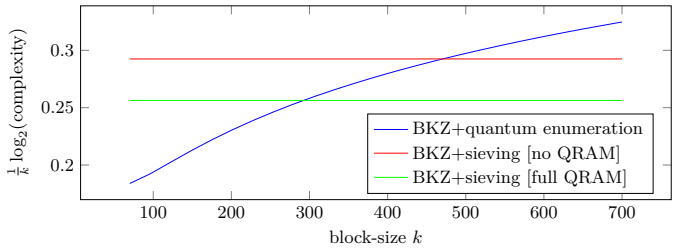


Fig. 1: Comparison between BKZ using quantum enumeration with full quadratic speedup, and BKZ using our lower bound on quantum sieving with no QRAM. We also include the best quantum algorithm with no constraints on the QRAM.

Symmetric key cryptography. The idea of designing quantum algorithms with a small QRAM can be applied to the collision finding and multi-target preimage search problems, discussed in [19] without QRAM. We demonstrate a smooth trade-off for the quantum collision finding problem between the results of [17] and [19], which correspond to the maximal and minimal QRAM, respectively. We also show a similar trade-off for the multi-target preimage search problem bridging the QRAM-based multi-target Grover algorithm and the result in [19].

These results readily apply to the applications discussed in [19]: hash collision finding, multi-user security, CBC mode of operations, as well as building blocks for advanced cryptanalysis such as [39]. As in the lattice cryptanalysis discussion, small practical QRAM helps for symmetric-key cryptanalysis but the same caveat applies regarding the need for efficient data replacement.

Quantum lattice sieving without QRAM. Finally, we present, inspired by [33], a quantum sieving algorithm without QRAM that has a depth complexity smaller than the classical sieving algorithms. This does not contradict our lower bound because it uses exponentially many qubits to operate many gates in parallel, which may be considered unrealistic; when $2^{0.207d}$ qubits are available, our algorithm runs in time about $2^{0.279d}$. To the best of our knowledge, this is the first QRAM-less quantum algorithm in the standard circuit model, faster than classical, with the minimal space of $2^{0.207d}$ for lattice sieving.[5] This result rules out a depth-alone lower bound for sieving, suggesting that the depth-width cost could be more desirable than the depth cost.

[5] [41] suggested a fast quantum algorithm without QRAM, but their algorithm works in a stronger model of distributed quantum computing [11].

Organization. This paper is organized as follows. Section 2 presents some preliminaries required for this paper. We present the quantum algorithms using small QRAM in Section 3. As applications of these algorithms, the trade-off algorithms between the QRAM size and time complexity are given in Section 4 for lattice sieving and in Section 5 for symmetric key cryptography. Our lattice sieving model and lower bounds are presented in Section 6, with a more formal treatment in Appendix B. Finally, Section 7 shows how to solve the sieving problem without a QRAM but at the expense of using many qubits.

2 Preliminaries

Notations. We use bold lower-case letters to denote vectors and bold upper-case letters to denote matrices. The Euclidean norm of a vector $\boldsymbol{v}$ is denoted by $\|\boldsymbol{v}\|$. The inner product of two vectors is denoted by $\langle \cdot, \cdot \rangle$. We usually denote the dimension of the lattice by d, and also assume that the size of the vectors are polynomial in d.

When discussing the running time of the algorithms solving lattice problems, we assume that the vectors (used in the basis or in computations) have bit-size polynomial in the ambient dimension d). In particular, we will ignore the resulting polynomial factors due to manipulating vectors.

2.1 Lattice

A (full-rank) *d-dimensional lattice* $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^d$, whose elements can be uniquely expressed as the linear combinations of linearly independent basis vector $\mathbf{B} = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_d\}$.

$$\mathcal{L}(\mathbf{B}) = \left\{ \sum_{i=1}^{d} x_i \cdot \boldsymbol{b}_i \mid x_i \in \mathbb{Z}, \boldsymbol{b}_i \in \mathbf{B} \right\} \tag{1}$$

Given a basis $\mathbf{B}$ of the lattice, the *shortest vector problem (SVP)* asks to find the vector $\boldsymbol{s} \in \mathcal{L}(\mathbf{B})$ such that $\|\boldsymbol{s}\| = \lambda_1(\mathcal{L}) := \min_{\boldsymbol{v} \in \mathcal{L}(\mathbf{B}) - \{\mathbf{0}\}} \|\boldsymbol{v}\|$.

2.2 Quantum computing

We consider the quantum circuit model consisting of single- and two-qubit gates without any locality constraint. We further assume that the number of qubits available to algorithms is bounded by some polynomial, except for Sections 6 and 7 which focus on theoretic perspectives.

We count the depth of the quantum circuit as the time complexity. We occasionally ignore the polynomial factors and only focus on the exponential terms when discussing the complexity. Since this paper mostly assumes a small number of qubits, choosing different complexity measures does not change the results of this paper much. We explicitly describe the number of qubits when we discuss algorithms with large qubits. For a more detailed introduction to quantum computing, we refer to [50].

Quantum oracles. Let X be a set. For a function $f : X \to \{0,1\}^m$, the oracle O_f is a unitary that computes:

$$O_f : |x, y\rangle \mapsto |x, y \oplus f(x)\rangle . \tag{2}$$

The circuits for computing most oracles in this paper will be explicitly given. The time complexity of the oracle is the same as the corresponding circuit.

For a projector P acting on the span of X, we similarly define the projection oracle O_P by:

$$O_P : |\psi\rangle |b\rangle \mapsto \begin{cases} |\psi\rangle |b \oplus 1\rangle & \text{if } |\psi\rangle \in Im(P), \\ |\psi\rangle |b\rangle & \text{if } |\psi\rangle \in Ker(P). \end{cases} \tag{3}$$

We occasionally use quantum amplitude amplification as a subroutine in our algorithms, which can be seen as a generalization of Grover's algorithm.

Theorem 1 (Quantum amplitude amplification [16]). *Let P be a projector acting on the span of X. Let $Init$ be a quantum algorithm that generates $|\phi\rangle = \alpha |\phi_P\rangle + \beta \left|\phi_P^{\perp}\right\rangle$, where $|\phi_P\rangle \in Im(P)$ and $\left|\phi_P^{\perp}\right\rangle \in Ker(P)$. Let $\theta \in [0, \pi/2]$ be such that $\sin\theta = |\alpha|$. Let $N = \lfloor \frac{\pi}{4\theta} - \frac{1}{2} \rfloor$. If the time complexities of O_P and $Init$ are T_P, T_{Init}, respectively, then there exists a quantum algorithm that produces a quantum state sufficiently close[6] to $|\phi_P\rangle$ in time $O(N(T_P + T_{Init}))$. If $\alpha = o(1)$, it can be written as $O((T_P + T_{Init})/\alpha)$.*

If $X = \{0,1\}^n$ and the projector P is defined by a function $f : \{0,1\}^n \to \{0,1\}$, we recover Grover algorithm. In particular, it finds $x \in \{0,1\}^n$ such that $f(x) = 1$ in time $O(\sqrt{2^n/|f^{-1}(1)|})$.

Theorem 2 (Quantum Minimum Finding Algorithm [25, Theorem 1]). *Given a list of $N = 2^n$ values from an ordered set, there is an algorithm to find the index of the minimum value with probability at least 1/2 in time $O(\sqrt{N})$.*

2.3 Quantum random access memory

We consider *quantum random access memory (QRAM)* in addition to the quantum circuit model. The main feature of the QRAM is that it allows queries that are superposed in addresses, thereby allowing the superposition of desired data stored in memory.

More precisely, using the QRAM storing m elements $L = (x_1, ..., x_m)$, a quantum algorithm can apply a special QRAM gate that works as follows in the computational basis:

$$U_{QRAM}^{L} : |i, y\rangle \mapsto |i, y \oplus x_i\rangle . \tag{4}$$

The quantum algorithms in this paper are specified by a quantum circuit with single-, two-qubits and QRAM gates along with the memory storing the data in the QRAM. We count the QRAM gate as a single gate in the time complexity.

[6] This can be done in a standard way, e.g., as in [15].

We assume that the memory used in the QRAM gate always stores classical data. Precisely, we only consider *quantum random access classical memory (QRACM)*. One may consider quantum random access quantum memory (QRAQM), which is more general [38]. Given that maintaining a coherent state for a long time is hard, restricting the algorithms to QRACM only and having small quantum memory is a reasonable model: The small quantum memory is the only part that maintains quantum information perpetually, while the other parts only become coherent ephemerally.

Although there is no agreed model of the QRA(C)M, writing data on it will require non-trivial cost which is at least logarithmic in the memory size (and more likely to require polynomial or exponential time). However, we assume that writing the data to, or overwriting (i.e. reset and load another data) the QRACM can be done in time $O(1)$. While this assumption is very strong, the complexity assuming this assumption gives a lower bound of the complexity of the algorithm, in the sense that the cost related to memory I/O is excluded, and only the number of memory accesses is taken into account. Also, the result yields a natural trade-off for some quantum algorithms using QRAM, as can be seen in Section 4. More discussion on this assumption can be found in Section 4.4 where we argue that our model is not as unrealistic as it appears, if we think about QRAMs as physical objects.

3 Quantum Algorithms with Bounded QRAM

This section presents the basic quantum algorithms using bounded-size QRAM. We suppose that a set $X = \{x_1, ..., x_M\}$ of size M is given to the algorithm in the memory, but the algorithm is allowed to use QRAM of size $S \leq M$, which means the algorithm can coherently access at most S elements in X. For simplicity, we assume that M is divided by S. Looking ahead, we are mainly interested in the case where X is a random subset of $\mathcal{S}^{d-1}$ for later use in lattice sieving, but the results in this section hold regardless of the choice of X.

Consider the function $f : X \to \{0, 1\}$ and the problem of finding $x \in X$ such that $f(x) = 1$[7]. When $X = [M]$ is explicitly given and the function f can be computed by a circuit (or given as an oracle), it is well known that Grover algorithm requires only $O(\sqrt{M})$ computations of f to solve this problem [30].

The situation becomes more complicated if X is not explicitly specified a priori. To execute the quantum search or the amplitude amplification, we need to implement the reflection map

$$\mathrm{Ref} = I - 2\left|\psi\right\rangle\!\left\langle\psi\right| \text{ for } \left|\psi\right\rangle = \sum_{i\in[M]} \frac{\left|i, x_i\right\rangle}{\sqrt{M}}. \tag{5}$$

[7] With the promise that such an x exists with a high probability.

If there is no size bound on the QRAM, an algorithm that outputs $|\psi\rangle$ can be efficiently implemented as follows

$$|0\rangle \mapsto \sum_{i\in[M]} \frac{|i,0\rangle}{\sqrt{M}} \mapsto \sum_{i\in[M]} \frac{|i,x_i\rangle}{\sqrt{M}} \tag{6}$$

where we use the QRAM gate (eq. (4)) in the second map, and the amplitude amplification gives the same complexity as in Grover's algorithm. When the algorithm is limited to a QRAM of size at most S, the following lemma shows that we can obtain a slightly worse quantum speedup. Setting $S = 1$ and $S = M$ give the classical exhaustive search algorithm and Grover algorithm, respectively.

Lemma 1. *Let X be a set of size M stored in a classical memory of the algorithm. Assume that the function $f : X \to \{0,1\}$ is randomly chosen so that $\Pr[f(x) = 1] = c/M$ holds for each $x \in X$ independently for some $c \geq 6$.*[8] *If QRAM of size S is available for some $0 \leq S \leq M$, then there exists an algorithm A that can find x^* such that $f(x^*) = 1$ in $O\left(\frac{M}{\sqrt{S}}\right)$ evaluations of f with a sufficiently high probability (say 0.99).*

Proof (sketch). The proof idea is that by storing S elements of X in the QRAM and searching the solution in those elements only using $O(\sqrt{S})$ evaluations. Repeating the procedure M/S times for each block, $O\left(\frac{M}{\sqrt{S}}\right)$ evaluations are required. The formal proof is given in Appendix A. □

We remark that the assumption for the efficient replacement of QRAM is crucial in the above lemma. Otherwise, the first step to store elements in QRAM can dominate the algorithm's complexity.

We also consider the following generalization of the search problem: Given two sets X and Y and a function $f : X \times Y \to \{0,1\}$, find most solution pairs $(x,y) \in X \times Y$ such that $f(x,y) = 1$. Lemma 2 shows the complexity of this problem in the bounded QRAM case.

Lemma 2. *Let X, Y be a set of sizes M_1, M_2 respectively, stored in a classical memory of the algorithm. Assume that the function $f : X \times Y \to \{0,1\}$ has K (uniformly distributed) solutions $(x,y) \in X \times Y$ such that $f(x,y) = 1$. If two QRAMs of size S each are available for some $0 \leq S \leq \max(M_1, M_2)$, then there exists an algorithm A' that can find $\Omega(K)$ solutions using*

1. *$O\left(\sqrt{M_1 \cdot M_2 \cdot K}\right)$ evaluations of f if $\sqrt{\frac{M_1 \cdot M_2}{K}} \leq S \leq \max(M_1, M_2)$ and*
2. *$O\left(\frac{M_1 \cdot M_2}{S}\right)$ evaluations of f if $1 \leq S \leq \sqrt{\frac{M_1 \cdot M_2}{K}}$.*

Proof (sketch). By storing S elements of X and Y in separate QRAMs, superposition of S^2 possible pairs can be generated, and the amplitude amplification can be applied, similar to the proof in Lemma 1. The result slightly differs because we need to find $\Omega(K)$ solutions. Detailed proof is given in Appendix A. □

[8] It ensures the existence of x^* such that $f(x^*) = 1$ with probability at least 0.995.

We also prove the lower bound corresponding to Lemma 1, albeit in a slightly restricted model. The condition "each query contains at most S elements" captures the situation that whenever A makes a QRAM query, A evaluates the f information of the QRAM entries. The proof is more involved and uses a variant of the compressed oracle argument [59] for the Bernoulli random functions [23]. We defer the proof to Appendix A.1.

Theorem 3. *Let X be a set of size M. Suppose that $f : X \to \{0,1\}$ be a random function where $f(x) = 1$ holds with probability $p = \Omega(1/M)$ for each $x \in X$ independently. Let A be a quantum algorithm that makes at most q queries to the quantum oracle access to f. If each query of A to f contains at most S elements in X in the computational basis, then it holds that*

$$\Pr\left[A^f \to x : f(x) = 1\right] = O\left(\sqrt{S} \cdot pq\right).$$

In particular, any algorithm finding the solution x requires making at least $\frac{M}{\sqrt{S}}$ queries to f for $p = \Theta(1/M)$.

4 Time-QRAM Trade-off for Quantum Lattice Sieving

This section presents the fine-grained trade-off between time and the QRAM size in quantum lattice sieving based on the algorithms in Section 3. We focus on the locality-sensitive filtering(LSF) method in [12], which is proven to be optimal among a wide class of sieving algorithms in the classical setting [40].

Locality-sensitive filtering defines a family of functions called *filters* $\{\mathcal{F}_i\}_{i\in[t]}$. A vector $\boldsymbol{v}$ *passes* a filter $\mathcal{F}_i$ if (say) $\mathcal{F}_i(\boldsymbol{v}) = 1$, otherwise $\mathcal{F}_i(\boldsymbol{v}) = 0$. With careful construction of filter family, the key part of LSF is that one can find all filters that a vector passes, without calculating every $\mathcal{F}_i(\boldsymbol{v})$ for $i \in [t]$.

4.1 Sieving with locality-sensitive filtering

Before going into the quantum algorithm, we review the classical lattice sieving algorithm with locality-sensitive filtering in Algorithm 1.

Heuristic lattice sieving. [49] proves that $|L| = (\frac{4}{3})^{d/2+o(d)}$ suffices to heuristically solve the shortest vector problem in the lattice of dimension d. Throughout this paper, we fix the size of the input list as

$$n := |L| \approx 2^{0.2075d+o(d)}. \tag{7}$$

We call that the vector $\boldsymbol{v} - \boldsymbol{w}$ added in Line 4 the *reduced vector*. The pair $(\boldsymbol{v}, \boldsymbol{w})$ is called by the *reducing* pair. We always consider $R'/R \to 1$ (as $d \to \infty$). In this case, the condition $\|\boldsymbol{v} - \boldsymbol{w}\| \leq R'$ is roughly equivalent to the angle between them is less than $\pi/3$, which we focus on. The time complexity is dominated by the time to find the close pairs in Line 3. In the original heuristic sieve algorithm [49], it is done by the exhaustive search on $(\boldsymbol{v}, \boldsymbol{w}) \in L \times L$, thereby the time complexity becomes $n^2 \approx 2^{0.4150d+o(d)}$. Hash-based approaches significantly reduce the time complexity, and we review the most efficient algorithm based on the locality-sensitive filtering below.

Algorithm 1 Lattice Sieving Algorithms

Input $R, R' = \gamma \cdot R > 0$, L: list of input vectors with $\|\boldsymbol{v}\| \leq R \quad \forall \boldsymbol{v} \in L$
Output L': list of output vectors with $\|\boldsymbol{v}'\| \leq R' \quad \forall \boldsymbol{v}' \in L'$

$L' \leftarrow \emptyset$
for all $\boldsymbol{v} \in L$ **do**
  if $\exists \boldsymbol{w} \in L$ such that $\boldsymbol{v} \neq \boldsymbol{w}$ and $\|\boldsymbol{v} - \boldsymbol{w}\| \leq R'$ **then**
    $L' \leftarrow L' \cup \{\boldsymbol{v} - \boldsymbol{w}\}$
return L'

Geometry of sphere. Consider the unit sphere $\mathcal{S}^{d-1} := \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| = 1\}$ and half-(hyper)spaces $\mathcal{H}_{\boldsymbol{v},\alpha} := \{\boldsymbol{x} \in \mathbb{R}^d : \langle \boldsymbol{x}, \boldsymbol{v} \rangle \geq \alpha\}$. For $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{S}^{d-1}$ and $\alpha, \beta \in [0, 1]$, spherical caps and wedges are defined by

$$\mathcal{C}_{\boldsymbol{v},\alpha} := \mathcal{S}^{d-1} \cap \mathcal{H}_{,\alpha}, \quad \mathcal{W}_{\boldsymbol{v},\alpha,\boldsymbol{w},\beta} := \mathcal{S}^{d-1} \cap \mathcal{H}_{\boldsymbol{v},\alpha} \cap \mathcal{H}_{\boldsymbol{w},\beta}.$$

Let μ be the canonical Lebesgue measure over $\mathbb{R}^d$. The relative volumes of spherical caps and wedges have an important role in the lattice analysis. If the vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{S}^{d-1}$ satisfies $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = \cos\theta$, we define

$$\mathcal{C}_d(\alpha) := \frac{\mu(\mathcal{C}_{\boldsymbol{v},\alpha})}{\mu(\mathcal{S}^{d-1})}, \quad \mathcal{W}_d(\alpha, \beta, \theta) := \frac{\mu(\mathcal{W}_{\boldsymbol{v},\alpha,\boldsymbol{w},\beta})}{\mu(\mathcal{S}^{d-1})}.$$

We recall the estimates of these two values for large d [45,12].

Lemma 3. *For arbitrary constants $\alpha, \beta \in (0, 1)$ and $\theta \in [0, \pi]$, the following asymptotic formulas hold:*

1. $\mathcal{C}_d(\alpha) = \mathrm{poly}(d) \cdot \left(\sqrt{1 - \alpha^2}\right)^d$, *and*
2. $\mathcal{W}_d(\alpha, \beta, \theta) = \mathrm{poly}(d) \cdot \left(\sqrt{1 - \gamma^2}\right)^d$ *for* $\gamma = \sqrt{\frac{\alpha^2 + \beta^2 - 2\alpha\beta\cos\theta}{\sin^2\theta}}$.
   In particular, if $\alpha = \beta$, $\mathcal{W}_d(\alpha, \beta, \theta) = \mathrm{poly}(d) \cdot \left(\sqrt{1 - \frac{2\alpha^2}{1+\cos\theta}}\right)^d$.

Locality-sensitive filtering. The (spherical) locality-sensitive filtering is a family of functions called *filters* $\{\mathcal{F}_{i,\alpha}\}_{i \in [t], \alpha \in (0,1)}$ that are specified by a vector in a set $\{\boldsymbol{c}_1, ..., \boldsymbol{c}_t\}$ (sometimes called centers) in $\mathcal{S}^{d-1}$ along with the parameter $\alpha \in (0, 1)$. This family has a nice property that, given a vector $\boldsymbol{v} \in \mathcal{S}^{d-1}$ and $\beta \in (0, 1)$ as input, it is possible to find all indices $i \in [t]$ such that $\langle \boldsymbol{v}, \boldsymbol{c}_i \rangle \geq \beta$ efficiently. We call $\mathcal{F}_{i,\beta}$ by the relevant β-filters for these i's. Formally, if the number of relevant β-filters is $f_{\boldsymbol{v},\beta}$, then it finds all indices in time $O(f_{\boldsymbol{v},\beta})$. The construction of locality-sensitive filtering is based on the random product code, and the expected number of relevant filters is $t \cdot \mathcal{C}_d(\beta)$.

The result of classical lattice sieving can be summarized as follows. We include a sketch of the proof for the completeness of the paper.

Theorem 4 ([12, Theorem 7.1]). *Let L be a list of n random input vectors sampled from $\mathcal{S}^{d-1}$ where n is defined in (7). There is a classical lattice sieving algorithm, given L and parameters $\alpha, \beta \in (0,1)$ as input, that outputs $\Omega(n)$ reduced vectors based on the LSF with $t = \mathcal{W}_d(\alpha, \beta, \pi/3)^{-1}$ filters. The expected running time of the algorithm is given as follows:*

$$T = \underbrace{nt \cdot \mathcal{C}_d(\beta)}_{\text{fill } \beta\text{-filters}} + n \cdot \left(\underbrace{t \cdot \mathcal{C}_d(\alpha)}_{\text{find } \alpha\text{-close filters}} + \underbrace{nt \cdot \mathcal{C}_d(\alpha) \cdot \mathcal{C}_d(\beta)}_{\text{examine vectors in } \alpha\text{-close filters}} \right). \quad (8)$$

In particular, for the optimal choice, the time complexity becomes $2^{0.2925d+o(d)}$.

Proof (sketch). Divide $L = C \cup (L \setminus C)$ into two lists of similar sizes. We first describe the algorithm A as follows.

1. Prepare the spherical locality-sensitive filtering of size t defined by a set $\{\boldsymbol{c}_1, ..., \boldsymbol{c}_t\}$. Define t empty lists $B_1, ..., B_t$. Define an empty list L'.
2. For each vector $\boldsymbol{w} \in C$, find all $i \in [t]$ such that $\mathcal{F}_{i,\beta}$ is the relevant β-filters of $\boldsymbol{w}$, and append $\boldsymbol{w}$ to B_i.
3. For each $\boldsymbol{v} \in L \setminus C$, find all $i \in [t]$ and $\boldsymbol{w} \in B_i$ such that $\mathcal{F}_{i,\alpha}$ is the relevant α-filters of $\boldsymbol{v}$, and check if $\|\boldsymbol{v} - \boldsymbol{w}\| \leq R'$. If true, append $\boldsymbol{v} - \boldsymbol{w}$ to L'.
4. Output L'.

We first analyze the time complexity of the algorithm A. Note that $|C|, |L \setminus C| = O(n)$. The first step can be done efficiently. For the second step, finding the relevant vectors can be done efficiently due to the definition of locality-sensitive filtering. Since the expected number of the relevant filters is $t \cdot \mathcal{C}_d(\beta)$, this step can be done in time $O(nt \cdot \mathcal{C}_d(\beta))$ at total. The third step is similar; finding relevant α-filters is done in time $t \cdot \mathcal{C}_d(\alpha)$ and each of B_i contains $n \cdot \mathcal{C}_d(\beta)$ vectors on expectation, giving the desired time complexity as in eq. (8).

For the correctness of the algorithm, consider a close vector pair $(\boldsymbol{v}, \boldsymbol{w})$, i.e., $\langle \boldsymbol{v}, \boldsymbol{w} \rangle \geq \cos(\frac{\pi}{3})$. The probability that a random vector $\boldsymbol{c}$ defines a relevant β- and α-filter of $\boldsymbol{v}$ and $\boldsymbol{w}$, respectively, is precisely computed by $\mathcal{W}_d(\alpha, \beta, \pi/3)$. Hence, by Lemma 3, choosing

$$t = \mathcal{W}_d(\alpha, \beta, \theta)^{-1} = \mathcal{W}_d\left(\alpha, \beta, \frac{\pi}{3}\right)^{-1} \approx \left(1 - \frac{4}{3}\left(\alpha^2 - \alpha\beta + \beta^2\right)\right)^{-d/2} \quad (9)$$

ensures that there exists a filter that can be used to find $\boldsymbol{v} - \boldsymbol{w}$ on expectation. Since $n = 2^{0.2075d+o(d)}$ ensures that the number of such elements is at least $\Omega(n)$, we conclude that the output L' of A contains $\Omega(n)$ vectors.

By setting the three terms to be equal to each other, the optimization results yield $\alpha = \beta = \frac{1}{2}$ and $T = t = \left(\frac{3}{2}\right)^{d/2+o(d)} = 2^{0.2925d+o(d)}$. □

4.2 Quantum search inside the α-close filters

Applying a quantum search algorithm for searching close vectors in B_i, [42] obtains quantum sieving in time $2^{0.2653d+o(d)}$.

Theorem 5 ([42, Section 14.2.10]). *There is a quantum lattice sieving algorithm, given a list of n random input vectors and $\alpha, \beta \in (0,1)$, that outputs $\Omega(n)$ reduced vectors using the quantum LSF with $t = \mathcal{W}_d(\alpha, \beta, \pi/3)^{-1}$ filters. The running time of the algorithm is*

$$T_1 = nt \cdot \mathcal{C}_d(\beta) + nt \cdot \mathcal{C}_d(\alpha) + n \cdot [nt \cdot \mathcal{C}_d(\alpha) \cdot \mathcal{C}_d(\beta)]^{1/2}. \tag{10}$$

In particular, the optimal choice gives the time complexity of $2^{0.2653d+o(d)}$.

Proof. The first two terms are derived in exactly the same way as in Theorem 4. When finding the reducing pair for the query vector $\boldsymbol{v}$ in the union D_v of the B_i such that B_i is α-close to v, quantum amplitude amplification is used with the oracle function

$$F_{\boldsymbol{v}}(\boldsymbol{w}) = 1 \text{ iff } \|\boldsymbol{v} - \boldsymbol{w}\| \leq R' \tag{11}$$

along with the reflection on the uniform superposition of the vectors in D_v. Since the expected size of the search space is $(n \cdot \mathcal{C}_d(\alpha)) \cdot (t \cdot \mathcal{C}_d(\beta))$, we obtain the desired time complexity.

Letting the three terms be equal to each other, we get $\alpha = \beta = \frac{\sqrt{3}}{4} = 0.4330$ and $T_1 = \left(\frac{13}{9}\right)^{d/2+o(d)} = 2^{0.2653d+o(d)}$. □

To implement the reflection operator, all of $S = nt \cdot \mathcal{C}_d(\alpha) \cdot \mathcal{C}_d(\beta)$ vectors in relevant B_i's should be stored in QRAM. With the parameters used in the optimized setting, the $S = 2^{0.1155d+o(d)}$ size QRAM is required. The trade-off relation between the size of QRAM and time can be induced using Lemma 1. If the allowed QRAM is bounded, we can get the time complexity as follows. For convenience, we denote the size of QRAM as γ^d, instead of 2^s in Lemma 1.

Theorem 6. *The time complexity of algorithm in Theorem 5 with a QRAM of size γ^d for $1 \leq \gamma \leq 13/12$ is*

$$T_2 \;=\; nt \cdot \mathcal{C}_d(\beta) + nt \cdot \mathcal{C}_d(\alpha) + n^2 t \cdot \mathcal{C}_d(\alpha) \cdot \mathcal{C}_d(\beta)/\gamma^{d/2} \tag{12}$$

For the optimal choice of α and β, we have $T_2 = \left(\frac{3\gamma}{3\gamma-1}\right)^{d/2+o(d)}$.

Proof. After finding filters α-close to the query vector, one uses Grover search over the list of vectors in the filter buckets with the expected number of entries being $(n\mathcal{C}_d(\alpha)) \cdot (t\mathcal{C}_d(\beta))$. The oracle function for the query vector $\boldsymbol{v}$ is $F_{\boldsymbol{v}}$ defined as eq. (11). With Lemma 1, a QRAM of size γ^d gives a $\gamma^{d/2}$ improvement in time, resulting in eq. (12). The size γ^d is bounded by the size of search space $nt \cdot \mathcal{C}_d(\alpha) \cdot \mathcal{C}_d(\beta)$, giving the bound of γ.

We can optimize the running time by equalizing the three terms. This gives $\alpha = \beta = \sqrt{1 - \frac{3\gamma}{4}}$ and $T_2 = \left(\frac{3\gamma}{3\gamma-1}\right)^{d/2}$ as stated. □

This algorithm can be improved by searching the reducing pairs within each filter bucket rather than with respect to buckets corresponding to each query vector, as shown in [18]. Although its optimal time complexity remains the same, the required QRAM is reduced by halving the exponent of γ^d.

Theorem 7. *There is a quantum lattice sieving algorithm that solves the problem in Theorem 6 with a QRAM of size γ^d with time complexity:*

$$T_3 = \begin{cases} nt \cdot \mathcal{C}_d(\beta) + nt \cdot \mathcal{C}_d(\alpha) + n^2 t \cdot \mathcal{C}_d(\alpha) \cdot \mathcal{C}_d(\beta)/\gamma^d \\ \quad \text{if } 1 \le \gamma^d \le \left[nt \cdot \mathcal{C}_d(\alpha) \cdot \mathcal{C}_d(\beta)\right]^{1/2} \\ nt \cdot \mathcal{C}_d(\beta) + nt \cdot \mathcal{C}_d(\alpha) + n \cdot \left[nt \cdot \mathcal{C}_d(\alpha) \cdot \mathcal{C}_d(\beta)\right]^{1/2} \\ \quad \text{if } \gamma^d > \left[nt \cdot \mathcal{C}_d(\alpha) \cdot \mathcal{C}_d(\beta)\right]^{1/2} \end{cases} \tag{13}$$

For an optimal choice of α and β, the time complexity is $T_3 = \left(\frac{3\gamma^2}{3\gamma^2-1}\right)^{d/2}$ for $1 \le \gamma \le \sqrt{\frac{13}{12}}$, and $T_3 = \left(\frac{13}{9}\right)^{d/2+o(d)} = 2^{0.2653d+o(d)}$ for $\gamma > \sqrt{\frac{13}{12}}$.

Proof. In contrast to previous theorems, filter buckets are filled with β-close center vectors and α-close query vectors in $nt \cdot \mathcal{C}_d(\beta) + nt \cdot \mathcal{C}_d(\alpha)$ time. We separate each with $B_{\mathcal{F},\beta}$ and $B_{\mathcal{F},\alpha}$ respectively, and each bucket stores $n\mathcal{C}_d(\beta)$ center vectors and $n\mathcal{C}_d(\alpha)$ query vectors respectively. The oracle function $F_{\mathcal{F}}$ for Grover search is defined as follows.

$$F_{\mathcal{F}}(\boldsymbol{v}, \boldsymbol{w}) = 1 \text{ iff } \|\boldsymbol{v} - \boldsymbol{w}\| \le R' \tag{14}$$

As explained in the proof of Theorem 4, the algorithm will output an expected $\Omega(n)$ reduced vectors, and therefore we expect $\Omega(n/t)$ reducing pairs in each filter on average, by the uniform randomness of vectors and filters. Using Lemma 2, the number of queries to find all $\Omega(n/t)$ solutions in each filter is

$$T_{\text{per filter}} = \begin{cases} n\mathcal{C}_d(\alpha) \cdot n\mathcal{C}_d(\beta)/\gamma^d \\ \quad \text{if } 1 \le \gamma^d < \left[n\mathcal{C}_d(\alpha) \cdot n\mathcal{C}_d(\beta)/(n/t)\right]^{1/2}, \\ \left[n\mathcal{C}_d(\alpha) \cdot n\mathcal{C}_d(\beta) \cdot (n/t)\right]^{1/2} \\ \quad \text{if } \left[\left(n\mathcal{C}_d(\alpha)\right) \cdot \left(n\mathcal{C}_d(\beta)\right)/(n/t)\right]^{1/2} < \gamma^d. \end{cases} \tag{15}$$

Multiplying eq. (15) by t gives the third term of T_3 for both cases. Optimizing the three terms, we obtain the result. The range of γ comes from the condition of γ^d separating the two cases. □

In conclusion, using Grover search to buckets with respect to each filter reduces the QRAM size, achieving $T_3 = 2^{0.2653d+o(d)}$ with only $2^{0.05778d+o(d)}$ QRAM. It can be easily checked that the result given in [18, Fig. 5] coincides with the relation in Theorem 7.

4.3 Quantum search over the α-close filters

[33] further improves the time complexity of quantum LSF by using Grover search to find a candidate bucket that contains a vector forming a reducing pair with the vector currently being processed. This gives a quadratic speedup on the second term of Theorem 4.

Lemma 4 ([33, Section 4]). *Let $\mathcal{F}_1, \ldots, \mathcal{F}_t$ be a list of LSF filters constructed via random product codes, and $\alpha \in (0,1)$. There is a randomized (classical) algorithm that given a vector $\boldsymbol{w}$, returns a uniformly random (pseudo[9]) α-close filter to $\boldsymbol{w}$ in time $\mathrm{poly}(d)$. This sampler requires $2^{o(d)}$ preprocessing time. The algorithm only requires $O(\log(d) \cdot \log(S(\boldsymbol{w}))$ random coins to output a filter on any given input $\boldsymbol{w}$, where $S(\boldsymbol{w})$ is the number of α-close filters to $\boldsymbol{w}$.*

Proof. Almost everything is a direct consequence of [33, Section 4.5]. The only nontrivial part is estimating the number of random coins which is not done in the analysis of [33]. The sampler works by constructing a tree in the pre-processing phase and pre-computing some values using dynamic programming. To sample a (pseudo) good α-filter, the algorithm simply goes down one branch of the tree randomly. At each node, it sample random an integer ℓ between 1 and "$\mathcal{L}^R(x)$" which is the number of "good leaves" in the tree (after the R-discretization). Furthermore, [33, Section 4.5.3] shows that if a filter is pseudo α-close then it is $(\alpha - \varepsilon)$-close where $\varepsilon = O(\frac{1}{d})$. Therefore, for large enough d, the number of pseudo α-close filters is essentially the same as the number of α-close filters. It follows that the algorithm only needs $\log_2(S)$ random coins at each level of the tree where S is the number of α-close filters. Finally, there are only $O(\log d)$ levels in the tree. □

Theorem 8 ([33]). *There is a quantum lattice sieving algorithm that given a list L of $O(n)$ random input vectors, uses quantum LSF with $t = \mathcal{W}_d(\alpha, \beta, \pi/3)^{-1}$ filters and parameters α, β to output $\Omega(n)$ reduced vectors in time*

$$T_4 = nt \cdot \mathcal{C}_d(\beta) + n\left[t \cdot \mathcal{C}_d(\alpha) + nt \cdot \mathcal{C}_d(\alpha) \cdot \mathcal{C}_d(\beta)\right]^{1/2}. \tag{16}$$

In particular, for an optimal choice of α and β, the complexity is $2^{0.2571d+o(d)}$.

Proof. In Theorems 5 to 7, filters which are α-close to a query vector are found classically in time $O\left(nt \cdot \mathcal{C}_d(\alpha)\right)$ with the help of the random product code (RPC). [33] explains how to obtain a sampler that can return a random (pseudo) α-close filter in $\mathrm{poly}(d)$ time. This sampler only requires $2^{o(d)}$ preprocessing time. By turning this sampler into a quantum circuit, we can get a superposition of α-close filters with respect to the query vector. Then the QRAM returns the β-close center vectors $\boldsymbol{w}$ for each filter f. Hence, for the query vector $\boldsymbol{v}$, we get the state

$$\sum_{\substack{i \\ \mathcal{F}_{i,\alpha} \text{is } \alpha\text{-close to } \boldsymbol{v}}} \sum_{\boldsymbol{w} \in B_i} |\mathcal{F}_{i,\alpha}\rangle |\boldsymbol{w}\rangle. \tag{17}$$

We can then apply Grover's algorithm with the oracle function $F_{\boldsymbol{v}}$ defined by

$$F_{\boldsymbol{v}}(\mathcal{F}_{i,\alpha}, \boldsymbol{w}) = 1 \text{ iff } \|\boldsymbol{v} - \boldsymbol{w}\| \leq R' \tag{18}$$

to obtain a quadratic speedup for the second term as well.

Optimization by letting the constituting terms equal to each other gives $\alpha = 0.4434, \beta = 0.5$, thereby the complexity is $2^{0.2571d+o(d)}$. □

[9] See proof, a pseudo α-close vector is always $(\alpha - \varepsilon)$-epsilon for some $\varepsilon = O(\frac{1}{d})$.

As far as we know, $2^{0.2571d+o(d)}$ is the best complexity achievable by using only a QRAM. We can now apply Lemma 1 to Theorem 8 in order to get a time-QRAM trade-off relation.

Theorem 9. *The time complexity of algorithm in Theorem 8 with a QRAM of size γ^d is*

$$T_5 = nt \cdot \mathcal{C}_d(\beta) + \left[nt \cdot \mathcal{C}_d(\alpha) + n^2 t \cdot \mathcal{C}_d(\alpha) \cdot \mathcal{C}_d(\beta)\right] / \gamma^{d/2}. \tag{19}$$

In particular, for an optimal choice of α and β, the time complexity is $T_5 = \left(\gamma - \frac{2}{3} + \frac{2}{3}\sqrt{1 - \frac{3}{4}\gamma}\right)^{-d/2}$ for $1 \le \gamma \le 1.07122$.

Proof. For each query $\boldsymbol{v}$, oracle function is defined as eq. (18). Therefore, Lemma 1 can be used for bounded QRAM, with the search space of $nt \cdot \mathcal{C}_d(\alpha) + n^2 t \cdot \mathcal{C}_d(\alpha) \cdot \mathcal{C}_d(\beta)$. Also, γ should satisfy $\gamma^d \le t \cdot \mathcal{C}_d(\alpha) + nt \cdot \mathcal{C}_d(\alpha) \cdot \mathcal{C}_d(\beta)$. Letting three terms equal to each other, the values of parameters are $\alpha = \sqrt{1 - \frac{3}{4}\gamma}$ and $\beta = 0.5$, giving $T_5 = \left(\gamma - \frac{2}{3} + \frac{2}{3}\sqrt{1 - \frac{3}{4}\gamma}\right)^{-d/2}$. These parameters also determine the range of γ by the constraint given above in this proof. □

In summary, by using a version of Grover search with bounded QRAM to the quantum sieving algorithm, we get an interpolation between classical LSF in [12] and the quantum LSF in [42,33]. Those results are summarized in Figure 2, showing the time complexity as a function of allowed QRAM size.

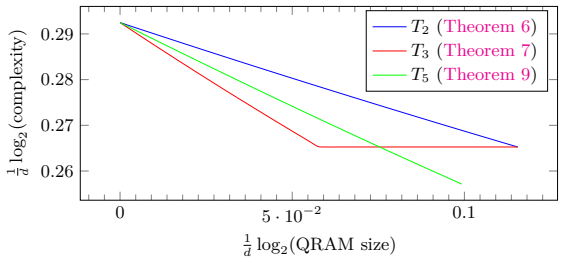


Fig. 2: Trade-off relations given in Theorems 6, 7 and 9. The top-left point represents the result of classical LSF [12]. The bottom-right point of each line represents the result of quantum LSF with no constraint on the QRAM size. In particular, we recover the results of Theorem 5 [42,18] (blue and red), and obtain a new trade-off from Theorem 8 [33] (green).

4.4 Discussion on the QRAM model

Recall that the trade-off obtained in Theorem 9 is under the QRAM model of Section 2.3. In particular, we assumed that writing the data to, or overwriting

(i.e. reset and load another data) the QRAM can be done in time $O(1)$. This assumption is clearly unrealistic but simplifies the proof to focus on the time-QRAM trade-off. Here, we present an alternative point of view to justify this assumption. In this alternative model, we replace the reload/overwrite operation above by two operations:

- Create: this operation loads N classical data into a QRAM of size N and takes time $O(N)$. Once created, the QRAM can be used as many times as wanted but the content cannot be changed anymore.
- Connect: this operation takes an already created QRAM and connects it to a quantum circuit so it can be used. This operation takes time $O(1)$.

The motivation behind this model is to view a QRAM as a physical object (say a quantum chip) where we somehow hardcode the data. Creating such an object is surely expensive and we assume takes time linear in the number of data stored. Once the object is created, it can be physically stored somewhere and, only when needed, connected to a quantum circuit to make I/O accesses. Connecting it to a circuit intuitively should not depend on the size of the QRAM. Importantly, in this model, it is entirely possible to create *many* distinct QRAMs.

Let us now see how this model applies to Theorem 9. In this algorithm, we need to store one list per filter (there are t filters). Each list contains $n \cdot \mathcal{C}_d(\beta)$ vectors on average. If we only limit ourselves to QRAMs of size γ^d then we need to create $nt \cdot \mathcal{C}_d(\beta)/\gamma^d$ QRAM in total. The cost of creating those QRAM is $n \cdot \mathcal{C}_d(\beta)$ which is equal to the first term of the complexity in Theorem 9 so in a certain sense, our complexity almost includes the cost of creating those QRAM. Note, however, that this will require to create and store an *exponential* number of QRAM. If we think about QRAM as physical objects, this might be difficult but not impossible. Later in the algorithm, we only ever use the QRAM initially created without any modifications. This means that we can replace the "reload" operations by a "connect" that runs in $O(1)$ and therefore obtain the same complexity.

In summary, this alternative model shows that our model is not as unrealistic as it appears if we think about QRAMs as physical objects. It of course still relies on the assumption that a QRAM can be implemented efficiently but this out of the scope of this paper.

5 Application of QRAM Trade-off to Symmetric Key

In Section 4, we introduced lattice sieving with a bounded QRAM based on Lemmas 1 and 2, which is the basis of public key cryptography. In this section, we show how to apply these methods to symmetric-key-related problems.

Quantum collision finding. The collision finding problem (CF) asks to find two elements $x_1, x_2 \in X$ such that $f(x_1) = f(x_2), x_1 \neq x_2$ given a (random) function $f : X \to Y$. It is known that $\Omega(N^{1/2})$, for $|X| = N$, is a lower bound on the

number of queries to f. With Grover search, [17] shows that $O(N^{1/3})$ queries are sufficient given access to a QRAM of size $O(N^{1/3})$. [19] gives an algorithm to solve the same problem with $O(N^{2/5})$ queries, but without using any QRAM. The main idea of quantum algorithms for CF is to precompute the values of f on some x's first, and then apply Grover or amplitude amplification for the rest of the data. While the algorithm in [17] stores data in a large QRAM, [19] stores them in a classical memory, and implements a search oracle with quantum gates. In other words, [19] converts the amount of QRAM into the time required to compare data one by one with gate operations. The time complexity of the algorithm in [19] is in eq. (20) below.

$$T = 2^{l+r/2} + 2^{(n-r-l)/2}\left(2^{r/2} + 2^{l}\right) \tag{20}$$

$$\left(|X| = 2^n,\ 2^l : \text{number of pre-calculated data},\ r \in [0, n]\ : \text{parameter}\right)$$

Optimizing the parameters gives $r = \frac{2n}{5}$ and $l = \frac{n}{5}$, resulting in $T = 2^{2n/5}$. The required classical memory is $2^l = 2^{n/5}$, which is also the cost of precalculating the data.

If a QRAM of size 2^γ is allowed, we can use the idea in Lemma 1 to obtain a speed up. We divide the pre-calculated data into blocks of size 2^s, load them into the QRAM, and access it as a membership query. This results in an algorithm of time complexity:

$$T' = 2^{l+r/2} + 2^{(n-r-l)/2}\left(2^{r/2} + 2^{l-\gamma}\right), \quad (0 \le \gamma \le l). \tag{21}$$

Optimizing the parameters gives $T' = 2^{2n/5-\gamma/5}$ with $2^{n/5+2\gamma/5}$ classical memory and a QRAM of size 2^γ. While this method has fewer effects on T than the outer parallelization introduced in [19, Section 5.3], it is still meaningful in that the approach is orthogonal to the outer parallelization; thereby, further improvement is possible by combining them. Also, outer parallelization needs $2^\gamma\times$ more qubits to achieve $2^{2n/5-3\gamma/5}$ complexity, while the above $2^{2n/5-\gamma/5}$ complexity with 2^γ QRAM causes only $O(n+\gamma)$ additional qubits.

Multi-target preimage search. Given a (random) permutation $H : X \to X$ and a set $T = \{y_1, \cdots, y_{2^t}\} \subset X$, the multi-target preimage search problem (MTPS) asks to find $i \in \{1, \cdots, 2^t\}$ and $x \in X$ such that $H(x) = y_i$. [19] also gives a quantum algorithm to solve the MTPS problem without using any QRAM. The time complexity is

$$T = 2^t + 2^{(n-t)/2}\left(2^{r/2} + 2^{t-r}\right), \qquad |X| = 2^n, r \in [0, n] : \text{parameter} \tag{22}$$

If we can freely choose all parameters, the optimization gives $t = \frac{3n}{7}$ and $r = \frac{2t}{3}$. However, t is usually a given parameter (i.e., the number of target images) in the MTPS problem, and the time complexity is expressed as a function t. If $t \ge \frac{3n}{7}$, then we can ignore some data to achieve the complexity of $2^{3n/7}$, while $2^{n/2-t/6}$ is the optimal complexity of [19] when $t < \frac{3n}{7}$.

With a QRAM of size 2^γ, the modified time complexity becomes

$$T' = 2^t + 2^{(n-t)/2}\left(2^{r/2} + 2^{t-r-\gamma}\right), \quad (0 \le \gamma \le t-r). \tag{23}$$

Optimizing the parameters gives $T' = 2^{3n/7-2\gamma/7}$ for $t \ge \frac{3}{7}n - \frac{2}{7}\gamma$, and $T' = 2^{n/2-t/6-\gamma/3}$ for $1 \le t < \frac{3}{7}n - \frac{2}{7}\gamma$. In both cases, the size of the QRAM is bounded by $2^\gamma \le 2^{n/3}$.

In summary, our strategy can be applied to the problems related to symmetric-key cryptography and can be applied to collision attacks on operation modes as discussed in [19].

6 Lower Bounds with Bounded QRAM

This section establishes the (conditional) lower bounds for the hash-based nearest-neighbor algorithms and lattice sieving in the bounded QRAM setting. Some parts of this section are adapted from [40].

We note that our lower bound is actually about the cryptographically relevant near-neighbor problems. To our knowledge, almost all sieving variants, such as the closest vector problem with preprocessing [24,43], use the near-neighbor subroutines; thereby, our lower bound applies. We refer to the discussion in [40] for the implication of the near-neighbor lower bounds.

6.1 The problems, models, and classical lower bounds

The near-neighbor problem. The lattice sieving algorithms usually maintain a list of lattice vectors, and find the close pairs in the list to construct a list of shorter vectors. Formally, the following problem is solved as a subroutine.

Definition 1 (Near-neighbor problem in $\mathcal{S}^{d-1}$). *Let $\theta \in [0, \pi]$. Let $L \subset \mathcal{S}^{d-1}$ be a finite subset of $\mathcal{S}^{d-1}$ whose elements are sampled uniformly at random from $\mathcal{S}^{d-1}$. In the near-neighbor problem, we 1) preprocess L in a certain data structure, and 2) later, when a uniformly random $\boldsymbol{x} \in \mathcal{S}^{d-1}$ is queried, we find almost all[10] vectors $\boldsymbol{y} \in L$ such that $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \ge \cos\theta$.*

The model for classical hash-based algorithms. In the high-dimensional case, the hash-based approaches have been known to be the most effective. Roughly, this approach divides the space into smaller regions using multiple random hash functions. In each of these *hash regions*, the algorithm searches for the close pairs, which is much more efficient than searching in the entire list. Formally, the hash-based near-neighbor search algorithms can be described as follows.

Definition 2 (Hash-based near-neighbor algorithms). *Given the near-neighbor problem parameterized by $L \subset \mathcal{S}^{d-1}$ and $\theta \in [0, \pi]$, the hash-based near-neighbor algorithm preprocesses the list L and processes queries $\boldsymbol{x}$ as given in Algorithm 2.*

Algorithm 2 The model of hash-based near-neighbor algorithms

SCHEME PARAMETERS:

- $t \in \mathbb{N}$ ▷ The number of hash regions
- $U_1, ..., U_t \subset \mathcal{S}^{d-1}$ ▷ Hash regions for insertions
- $Q_1, ..., Q_t \subset \mathcal{S}^{d-1}$ ▷ Hash regions for queries
- `method` $\in \{$`Query`, `FAS`$\}$ ▷ Choices for the finalization

Function INSERT($\boldsymbol{y}$)
  for all $i \in [t]$ **such that** $\boldsymbol{y} \in U_i$ **do**
    $B_i \leftarrow B_i \cup \{\boldsymbol{y}\}$

Function PREPROCESS(L)
  $B_1, ..., B_t \leftarrow \emptyset$
  for all $\boldsymbol{y} \in L$ **do**
    INSERT($\boldsymbol{y}$)

Function QUERY(L, θ)
  $P \leftarrow \emptyset$
  for all $\boldsymbol{x} \in L$ **do** ▷ For each $\boldsymbol{x} \in L$, find near neighbors $\boldsymbol{y}$.
    $C \leftarrow \emptyset$
    for all $i \in [t]$ **such that** $\boldsymbol{x} \in Q_i$ **do**
      for all $\boldsymbol{y} \in B_i$ **such that** $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \geq \cos\theta$ **do**
        $C \leftarrow C \cup \{\boldsymbol{y}\}$
    $P \leftarrow P \cup (\{\boldsymbol{x}\} \times C)$
  return P

Function FINDALLSOLUTIONS(L, θ)
  $P \leftarrow \emptyset$
  for all $i \in [t]$ **do** ▷ For each $i \in [t]$, find close pairs $(\boldsymbol{x}, \boldsymbol{y}) \in A_i \times B_i$.
    $C \leftarrow \emptyset$
    for all $(\boldsymbol{x}, \boldsymbol{y}) \in A_i \times B_i$ **such that** $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \geq \cos\theta$ **do**
      $C \leftarrow C \cup \{(\boldsymbol{x}, \boldsymbol{y})\}$
    $P \leftarrow P \cup C$
  return P

Function MAIN(L, θ)
  PREPROCESS(L)
  if `method` = `Query` **then** ▷ used in [40,33]
    $P \leftarrow$ QUERY(L, θ)
  else if `method` = `FAS` **then** ▷ used in [18,14]
    $A_1, ..., A_t \leftarrow \emptyset$
    for all $\boldsymbol{x} \in L$ **do** ▷ Preprocessing L regarding Q_i
      for all $i \in [t]$ **such that** $\boldsymbol{x} \in Q_i$ **do**
        $A_i \leftarrow A_i \cup \{\boldsymbol{x}\}$
    $P \leftarrow$ FINDALLSOLUTIONS(L, θ)
  return P

Our model has two methods `Query` and `FAS` for Main function. The first method `Query`, which was originally used in [40], searches for nearby vectors for each input vector using the function Query. On the other hand, the method `FAS` searches for pairs of close vectors in each hash region, reflecting the framework suggested in [18, Algorithm 1]. The difference between the two methods does not affect the lower bounds and proofs as described below.

Kirshanova and Laarhoven [40, Theorem 2 and 3][11] proved that choosing spherical caps of the same size for the hash regions gives the optimal algorithm. Following this, we assume that the hash regions are of the following form:

- Choose $\alpha, \beta \in [-1, 1]$ and draw $\boldsymbol{v}_i \leftarrow \mathcal{S}^{d-1}$ randomly for $i \in [t]$, and define

$$Q_i := \{\boldsymbol{z} \in \mathcal{S}^{d-1} : \langle \boldsymbol{z}, \boldsymbol{v}_i \rangle \geq \alpha\} \text{ and } U_i := \{\boldsymbol{z} \in \mathcal{S}^{d-1} : \langle \boldsymbol{z}, \boldsymbol{v}_i \rangle \geq \beta\}. \quad (24)$$

Query complexity. Most parts of computational cost in Algorithm 2 stem from the operations related to the input vectors in L and the filters. Here, we prove that the time complexity lower bound in [40] can be extended to a query lower bound regarding the operations for input vectors and filters, where the explicit definition of the query is as follows. A more formal definition requires the black-box model for the vectors and filters, which can be found in Appendix B.

Cost Model 1 (Classical query complexity)**.** *All operations related to the input vectors in L and filters are done through oracle access. Among these operations, the costs of the following queries are counted as the query complexity.*

1. *Given a vector $\boldsymbol{x}$, sampling a random index $i \in [t]$ such that $\boldsymbol{x} \in Q_i$ can be done at a unit cost; the filter corresponding to i is called by the* relevant *filter. If the number of relevant filters is $|Z|$, finding all of them can be done at $1 + |Z|$ unit costs.* [12] *This query is used in Lines 2 and 12. The algorithm can insert $\boldsymbol{x}$ in A_i or B_i corresponding to the relevant filter for free.*
2. *Given two vectors $(\boldsymbol{x}, \boldsymbol{y})$ as input, check if the inner product satisfies $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \geq 1/2$ or not at a unit cost. This query is used in Lines 13 and 21.*

The summation of the unit cost incurred by the above oracle queries during the algorithm is called the query complexity of the algorithm.

We can derive the following classical query lower bound by adapting [40] in this query model. The formal statement (Theorem 13) and proof of this theorem can be found in Appendix B along with a finer formalization.

Theorem 10 ([40, Theorem 4, adapted]). *The classical near-neighbor algorithm described by Algorithm 2 has query complexity at least $2^{0.2925d+o(d)}$, regardless of the choice of the method.*

[10] For example, we may ask to find 90% of the vectors as in [40, Definition 4].

[11] Strictly speaking, they showed the result below for `Query` method. Still, the collision probability parts (Theorem 2) are irrelevant to the method, and the equal choices for the caps (Theorem 3) are argued by looking at the overall complexity, which does not depend on the choice of method, as shown below.

[12] It corresponds to the assumption in the bottom of [40, p.6], which reflects the advanced hash-based algorithms. The constant $+1$ is to address the case of $|Z| = 0$.

6.2 The quantum time-memory trade-off lower bounds

Before proceeding to the quantum lower bound, we first introduce the model of quantum hash-based near-neighbor algorithms. We assume that the quantum algorithm also follows the framework given in Algorithm 2 and implements post-processing (Query or FindAllSolutions) quantumly with the same purpose, while the Preprocess part remains classical.

The known quantum algorithms indeed follow the same template with the quantum modifications for the procedure Query (e.g., in [33]) or FindAllSolutions (e.g., in [18,14]) in Algorithm 2.

The quantum speedup is from the procedures Query and FindAllSolutions. In particular, the vectors in the relevant filters B_i, i.e., the vectors $\boldsymbol{y} \in L$ such that $\boldsymbol{y} \in U_i$ are stored in the QRAM and coherently accessed during these procedures. Compared to the classical model, reading the input vectors in the filters or the relevant filter indices can be done coherently through the QRAM access. The quantum cost model can be summarized as follows; again, see Appendix B for a more detailed description.

Cost Model 2 (Quantum query/QRAM complexity)**.** *As in Cost Model 1, the operations related to the input vectors and filters can be done via oracle access. Among these, the following queries are counted in the complexity.*

1. *Given a vector $\boldsymbol{x}$, sampling a relevant filter can be done at a unit cost. Coherent access to some subset of the relevant filters can be done using the QRAM at a unit cost.*
2. *Given registers $\sum_{i,j} \alpha_{i,j} \left|\boldsymbol{x}_i, \boldsymbol{y}_j, r\right\rangle$ as input, compute the inner product $\sum_{i,j} \alpha_{i,j} \left|\boldsymbol{x}_i, \boldsymbol{y}_j, r + 1_{\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle \geq 1/2}\right\rangle$ in a unit cost where $1_{a \geq 1/2} = 1$ if $a \geq 1/2$, otherwise 0.*

Note that these queries require QRAM access. To formalize the usage of the QRAM, we introduce the following assumption on the coherent access to the input vectors, which roughly states that the coherent states of the input vectors are constructed only through the QRA(C)M access.

Assumption 1. *Let $V = \{\boldsymbol{v}_1, ..., \boldsymbol{v}_k\}$ be an arbitrary subset of the input list L. The coherent quantum state*

$$\sum_{i \in [k]} \alpha_i \left|i\right\rangle \left|\boldsymbol{v}_i\right\rangle \tag{25}$$

must be generated by the QRAM access to the list of k vectors.

One way to interpret Assumption 1 is to say that there is no efficient way to generate this superposition, except by using a QRAM. Indeed, it is clear that such a state can be generated by a plain quantum circuit, but all currently known ways of doing so are inefficient in some way, e.g., requires large depth

or qubits.[13] Indeed, if we implement the above state in the circuit model, the number of gates must be $\Omega(k)$ [38, Theorem V.2].

We also need a similar assumption on filter index access[14]. This operation is used, e.g., in Line 12 of the quantum version of Algorithm 2 [33].

Assumption 2. *Let $F = \{f_1, ..., f_k\}$ be an arbitrary subset of the set of the filter indices. The coherent quantum state*

$$\sum_{i \in [k]} \alpha_i |i\rangle |f_i\rangle \tag{26}$$

must be generated by QRAM access to the list of k vectors.

These assumptions imply that when the algorithm uses a QRAM of size 2^s, each register of the algorithm contains at most 2^s different vectors or filter indices. This is an important observation in the proof.

Quantum lower bound. Our quantum lower bounds for the bounded QRAM algorithms are summarized as follows.

Theorem 11. *The quantum near-neighbor algorithm described by Algorithm 2 with a QRAM of size 2^s has query complexity at least $2^{0.2925d-2s+o(d)}$ assuming Assumption 1 and Assumption 2.*

We describe the proof sketch here. The formal proof of the above theorem is given in Appendix B along with a further finer formalization of the model, which turns out to be quite technical.

Proof sketch. Our proof strategy is to simulate each query of the quantum lattice sieving (or near-neighbor) algorithm classically, inspired by [35]. Suppose that the quantum lattice sieving algorithm A, making Q quantum queries, uses a QRAM of size at most 2^s. For the quantum access to the input vectors in the relevant filter and indices, all the information possessed in these operations can be computed by 2^s corresponding classical queries in Cost Model 1. For the inner product query, note that each register has at most 2^s different vectors, so every inner product $\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$ can be computed by 2^{2s} classical inner product queries.

Then, we construct the *classical query* algorithm B whose output is identical to one of A. This is done by collecting all the information mentioned above

[13] One extreme is to sequentially read each vector, which requires a circuit of depth linear in the number of elements but only a few qubits. The other extreme is to build a tree (like in classical RAM) of depth logarithmic in the number of elements but whose width (i.e., the number of qubits) is linear in the number of elements. Some trade-offs between those two extremes are possible and suggest that one always needs either a very deep circuit or a large number of qubits

[14] In fact, the filter index itself can be coherently accessed without QRAM. However, whenever we want to check any information about the input vectors in the filter (e.g., checking if the filter is empty), it requires QRAM access.

classically. This simulation may take time[15] much longer than 2^d, but B always makes classical queries to the oracle, thereby obeying the classical query complexity lower bound in Theorem 10. Note that each QRAM query can be simulated by 2^s classical queries, and a quantum inner product query can be simulated by 2^{2s} classical inner product queries. This means that the classical query complexity of B is at most $2^{2s} \cdot Q$, which is lower bounded by $2^{0.2925d+o(d)}$. This establishes the lower bound in the statement. □

Plugging $s = 0$ gives the following noteworthy corollary.

Corollary 1. *There is no quantum speed-up for the lattice sieving problem without QRAM in our model under Assumption 1 and Assumption 2.*

7 Sieving Without QRAM

In this section, based on the algorithm in [33], we propose a quantum algorithm for lattice sieving without QRAM at the expense of using an exponential depth and number of qubits.

Lemma 5. *There is an algorithm that given t classical lists $B_1, \ldots, B_t$ of vectors, and in time $O(\sum_i |B_i|)$, creates a quantum circuit $\mathcal{O}$ that satisfies, for any vector $\boldsymbol{w}$ and any index $1 \leqslant i \leqslant t$:*

$$\mathcal{O} |i\rangle |\boldsymbol{w}\rangle |0\rangle \mapsto |i\rangle |\boldsymbol{w}\rangle |\boldsymbol{u}\rangle$$

where $\boldsymbol{u} = \arg\min_{\boldsymbol{x} \in B_i} \|\boldsymbol{x} - \boldsymbol{w}\|$ is the closest vector to $\boldsymbol{w}$ in the list B_i. The circuit $\mathcal{O}$ has depth $O(\max_i |B_i| + \log t)$, size $O(\sum_i |B_i|)$ and width $O(t)$.

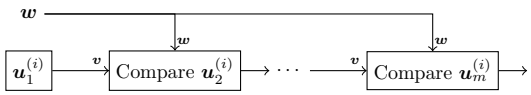
Proof. We will first build a classical circuit that performs this operation, and then convert it to a quantum circuit in the standard way.

Assume to simplify notations that each list B_i has exactly M vectors. Denote by $\boldsymbol{u}_1^{(i)}, \ldots, \boldsymbol{u}_M^{(i)}$ the vectors in B_i. The circuit is described in Figure 3. It uses two subcircuits:

- A standard multiplexer that takes t inputs $a_1, \ldots, a_t$ and an index i, and outputs a_i. This can be trivially implemented in depth $O(\log t)$, size $O(t)$ and width $O(t)$.
- A "Compare $\boldsymbol{u}$" circuit that takes two input vectors $\boldsymbol{v}$ and $\boldsymbol{w}$, and outputs $\arg\min_{\boldsymbol{x} \in \{\boldsymbol{u}, \boldsymbol{v}\}} \|\boldsymbol{x} - \boldsymbol{w}\|$. In other words, it returns the vector that is closest to $\boldsymbol{w}$ between the input $\boldsymbol{v}$ and the hardcoded vector $\boldsymbol{u}$. This can be implemented in constant depth, size, and width.

The idea of the circuit is very simple: for each i in parallel, the circuit computes the closest vector to $\boldsymbol{w}$ in B_i, and selects at the end the right one based on requested index. For a given i, the circuit sequentially computes the closest vector to $\boldsymbol{w}$. Specifically, one can show by induction on $1 \leqslant m \leqslant M$ that for any i and any $\boldsymbol{w}$, the output of the following circuit is $\arg\min_{j=1,\ldots,m} \|\boldsymbol{u}_j^{(i)} - \boldsymbol{w}\|$:

[15] Actual running time depends on the accuracy of the simulation.



Overall, the circuit has depth $O(M) + \log t$ since the longest path goes through M comparisons and the multiplexer. The width of the circuit is $O(t)$ since the t comparisons chains all happen in parallel. The size of the circuit is $O(Mt)$. It is clear that there is an algorithm that can construct this circuit is time at most $O(Mt)$ (the overall size). If the lists B_i have different sizes, the depth depends on the biggest B_i and the total size is the sum of the sizes of B_i.

We can convert this circuit to a quantum circuit using standard techniques. The resulting circuit has depth $O(M + \log t)$, size $O(Mt)$ and width $O(t)$. Furthermore, the algorithm that does the conversion runs in time linear in the size of the circuit.

Denote by $\mathcal{O}$ the quantum circuit obtained above where we add a register for the output. By the description of the circuit, it is clear that for any vector $\boldsymbol{w}$ and any index i,

$$\mathcal{O} \left|i\right\rangle \left|\boldsymbol{w}\right\rangle \left|0\right\rangle \mapsto \left|i\right\rangle \left|\boldsymbol{w}\right\rangle \left|\boldsymbol{u}\right\rangle$$

where $\boldsymbol{u} = \arg\min_{\boldsymbol{x} \in B_i} \|\boldsymbol{x} - \boldsymbol{w}\|$ is the closest vector to $\boldsymbol{w}$ in the B_i. □

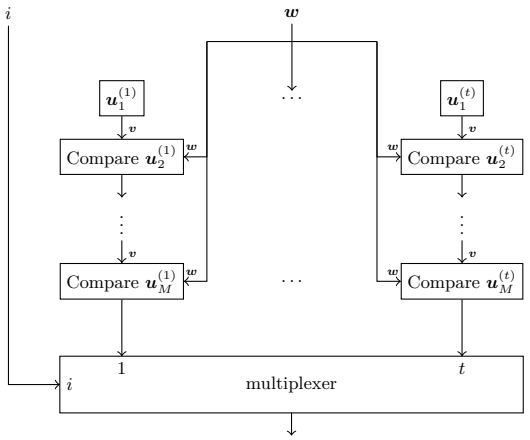


Fig. 3: Circuit diagram that outputs the vector in B_i closest to the input vector $\boldsymbol{w}$. Every vector in B_i is hardcoded in the circuit, and serially compared to $\boldsymbol{w}$. See Lemma 5.

Theorem 12. *There is a quantum lattice sieving algorithm that, given a list of n random input vectors and $\alpha, \beta \in (0, 1)$, outputs $\Omega(n)$ reduced vectors based on the LSF with $t = \mathcal{W}_d(\alpha, \beta, \pi/3)^{-1}$ filters. The running time of the algorithm is*

$$nt \cdot \mathcal{C}_d(\beta) + n \cdot \sqrt{t \cdot \mathcal{C}_d(\alpha)} \cdot \max(1, n \cdot \mathcal{C}_d(\beta)). \tag{27}$$

The algorithm uses no QRAM but evaluates a quantum circuit of depth $O(M + \log t)$*, size* $O(Mt)$ *and width* $O(t)$*.*

Proof. As in Section 4, the algorithm first prepare t LSF filters $\mathcal{F}_1, \ldots, \mathcal{F}_t$. For each i, we compute the classical list B_i of vectors inside $\mathcal{F}_i$.

We now apply Lemma 5 to build a quantum circuit $\mathcal{O}$ such that for any vector $\boldsymbol{w}$ and any index $1 \leqslant i \leqslant t$:

$$\mathcal{O} |i\rangle |\boldsymbol{w}\rangle |0\rangle \mapsto |i\rangle |\boldsymbol{w}\rangle |\boldsymbol{u}\rangle .$$

where $\boldsymbol{u} = \arg\min_{\boldsymbol{x} \in B_i} \|\boldsymbol{x} - \boldsymbol{w}\|$ is the closest vector to $\boldsymbol{w}$ in the list B_i. The circuit $\mathcal{O}$ has depth $O(\max_i |B_i| + \log t)$, size $O(\sum_i |B_i|)$ and width $O(t)$.

By Lemma 4, there is a (classical) sampler that can return, for any given $\boldsymbol{w}$, a random (pseudo) α-close filter to $\boldsymbol{w}$ in $\mathrm{poly}(d)$ time. This sampler only requires $2^{o(d)}$ preprocessing time. We can turn this sampler into a deterministic algorithm that takes as input R "random coins". Denote by $\mathcal{A}(\omega, \boldsymbol{w})$ such a run where ω denotes the random coins. By Lemma 4, we have $R = O(\log(d) \cdot \log(t \cdot C_d(\alpha)))$.

Consider the quantum oracle $\mathcal{O}'$ that on input $\omega \in \{0,1\}^R$ and $\boldsymbol{w}$:

- compute $i \leftarrow \mathcal{A}(\omega, \boldsymbol{w})$,
- returns $\mathcal{O}(i, \boldsymbol{w})$.

It is clear by the properties of $\mathcal{A}$ that $\mathcal{O}'(\omega, \boldsymbol{w})$ returns a vector $\boldsymbol{u} \in B_j$ for some j such that $\boldsymbol{w} \in B_j$. Overall, the sieving algorithm looks like this:

- Go through all n vectors and put each of them in the filters $\mathcal{F}_i$ to which they belong.
- Apply Lemma 5 to build $\mathcal{O}$.
- Use [33] to build $\mathcal{O}'$.
- For each vector $\boldsymbol{w}$ in the input list: run a quantum minimum finding algorithm using $\mathcal{O}'$ to find $\omega \in \{0,1\}^R$ that minimizes $\|\mathcal{O}'(\omega, \boldsymbol{w}) - \boldsymbol{w}\|$ and get the closest vector to $\boldsymbol{w}$ in the list. We then check if this vector forms a reduced pair and add it to the output list if that's the case. Based on the analysis done in the previous sections, we expect a constant number of vectors in the list to form a reduced pair with $\boldsymbol{w}$. Therefore by finding the closest vector to $\boldsymbol{w}$, we are sure to find a reduced pair using $\boldsymbol{w}$ if one exists.

In the first step, each vector belongs to $t \cdot \mathcal{C}_d(\beta)$ filters on average so the complexity is $nt \cdot \mathcal{C}_d(\beta)$. Furthermore, the lists B_i have an average size of $M = n \cdot \mathcal{C}_d(\beta)$.

In the second step, the complexity of building $\mathcal{O}$ is $O(\sum_i |B_i|) = O(Mt) = O(nt \cdot \mathcal{C}_d(\beta))$. The circuit $\mathcal{O}$ has depth $O(\max_i |B_i| + \log t) = O(M + \log t)$, size $O(Mt)$ and width $O(t)$.

In the third step, the complexity of building $\mathcal{O}'$ is dominated by the preprocessing cost of [33]'s sampler which is $2^{o(d)}$.

In the final step, the complexity is n times $2^{R/2}$ multiplied by the cost of the oracle $\mathcal{O}'$. The cost of $\mathcal{A}$ is $\mathrm{poly}(d)$ so this is essentially the cost of $\mathcal{O}$. Each evalutation of $\mathcal{O}$ costs the depth of its circuit which is $O(\max_i |B_i| + \log t) = O(M + \log t)$. The sampler $\mathcal{A}$ returns a uniform sample in the set of "good

filters" which is of average size $S = t \cdot \mathcal{C}_d(\alpha)$. Furthermore, $\mathcal{A}$ only requires $R = O(\log(d)\log(S))$ random coins. Since $\mathcal{A}$ is a uniform sampler in a set of size S, for a given $\boldsymbol{w}$, $\mathcal{A}(\cdot, \boldsymbol{w})$ takes each of the S possible outputs $2^R/S$ times. It therefore follows that $\mathcal{O}'(\cdot, \boldsymbol{w})$ takes each of the S possible output $k = 2^R/S$ times. As a result, running the minimum finding algorithm on $\mathcal{O}'(\cdot, \boldsymbol{w})$ takes time $O(\sqrt{2^R/k}) = O(\sqrt{S})$. Hence, the time complexity is

$$nt \cdot \mathcal{C}_d(\beta) + n \cdot \sqrt{t \cdot \mathcal{C}_d(\alpha)} \cdot \max(1, n \cdot \mathcal{C}_d(\beta)). \tag{28}$$

Here, the max is necessary to handle the case where there might be less than one vector in each bucket on average, because we still have to pay $O(1)$ just to examine the bucket. The number of qubits used is $O(t + R)$. □

We optimize the time complexity of Theorem 12 using the following formula:

$$t = \left(1 - \frac{4}{3}(\alpha^2 - \alpha\beta + \beta^2)\right)^{-d/2}, \quad \mathcal{C}_d(\alpha) = (1 - \alpha^2)^{d/2}, \quad n = (4/3)^{d/2}.$$

For each value of t, we compute the optimal time complexity by finding the optimal α and β such that t satisfies the equation above. Experimentally, we observe that the optimal complexity is close to

$$2^{0.414d - 0.655\log_2(t)} = \frac{2^{0.414d}}{t^{0.655}}$$

We observe a regime (see Figure 4) where we can achieve a better time complexity than the best classical sieving algorithm while using an exponential number of qubits. Note than we only draw the graph up to $t = 2^{0.207d}$ qubits, since there is a quantum algorithm in [41] that solves SVP in time $2^{0.1037d+o(d)}$ with $2^{0.207d+o(d)}$ qubits. [41] does not seem to work with less than $2^{0.207d+o(d)}$ qubits because it requires as many qubits as there are elements in the list, therefore it is incomparable with the above algorithm. Furthermore, the computational model of [41] is slightly different since it is more akin to a massively parallel/distributed system.

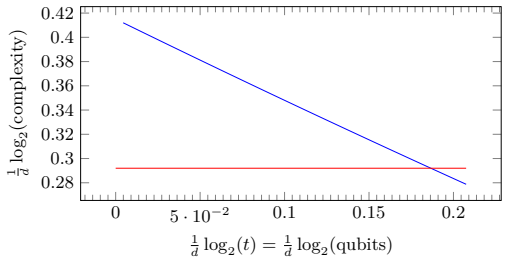


Fig. 4: Complexity of the sieve algorithm with no QRACM as a function of the number of qubits. The red line corresponds to the best classical complexity.

Acknowledgement

The work of Beomgeun Cho, Taehyun Kim, and Jeonghoon Lee was supported by Samsung Electronics Co., Ltd(IO221213-04119-01). The work of Yixin Shen was funded by the EPSRC grant EP/W02778X/2 and the France 2030 program managed by the French National Research Agency under grant agreement ANR-22-PETQ-0008 PQ-TLS.

References

1. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016), https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf
2. Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: Proceedings of the thirty-third annual ACM symposium on Theory of computing. pp. 601–610 (2001)
3. Albrecht, M.R.: On dual lattice attacks against small-secret lwe and parameter choices in helib and seal. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 103–129. Springer (2017)
4. Albrecht, M.R., Bai, S., Fouque, P.A., Kirchner, P., Stehlé, D., Wen, W.: Faster enumeration-based lattice reduction: Root hermite factor $k^{1/(2k)}$ time $k^{k/8+o(k)}$. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020. pp. 186–212. Springer International Publishing, Cham (2020)
5. Albrecht, M.R., Gheorghiu, V., Postlethwaite, E.W., Schanck, J.M.: Estimating quantum speedups for lattice sieves. In: Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26. pp. 583–613. Springer (2020)
6. Albrecht, M.R., Göpfert, F., Virdia, F., Wunderer, T.: Revisiting the expected cost of solving usvp and applications to lwe. In: Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23. pp. 297–322. Springer (2017)
7. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
8. Aono, Y., Nguyen, P.Q., Shen, Y.: Quantum lattice enumeration and tweaking discrete pruning. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 405–434. Springer (2018)
9. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber algorithm specifications and supporting documentation. NIST PQC Round **2**(4), 1–43 (2019)
10. Bai, S., van Hoof, M.I., Johnson, F.B., Lange, T., Ngo, T.: Concrete analysis of quantum lattice enumeration. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 131–166. Springer (2023)
11. Beals, R., Brierley, S., Gray, O., Harrow, A.W., Kutin, S., Linden, N., Shepherd, D., Stather, M.: Efficient distributed quantum computing. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences **469**(2153), 20120686 (2013)

12. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms. pp. 10–24. SIAM (2016)
13. Bindel, N., Bonnetain, X., Tiepelt, M., Virdia, F.: Quantum lattice enumeration in limited depth. In: Annual International Cryptology Conference. pp. 72–106. Springer (2024)
14. Bonnetain, X., Chailloux, A., Schrottenloher, A., Shen, Y.: Finding many collisions via reusable quantum walks: Application to lattice sieving. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 221–251. Springer (2023)
15. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte der Physik: Progress of Physics **46**(4-5), 493–505 (1998)
16. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. Contemporary Mathematics **305**, 53–74 (2002)
17. Brassard, G., Høyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: LATIN'98: Theoretical Informatics: Third Latin American Symposium Campinas, Brazil, April 20–24, 1998 Proceedings 3. pp. 163–169. Springer (1998)
18. Chailloux, A., Loyer, J.: Lattice sieving via quantum random walks. In: Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part IV 27. pp. 63–91. Springer (2021)
19. Chailloux, A., Naya-Plasencia, M., Schrottenloher, A.: An efficient quantum collision search algorithm and implications on symmetric cryptography. In: Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23. pp. 211–240. Springer (2017)
20. Chen, C., Danba, O., Hoffstein, J., Hülsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P., Whyte, W., Zhang, Z.: Algorithm specifications and supporting documentation. Brown University and Onboard security company, Wilmington USA (2019)
21. Cheon, J.H., Hhan, M., Hong, S., Son, Y.: A hybrid of dual and meet-in-the-middle attack on sparse and ternary secret lwe. IEEE Access **7**, 89497–89506 (2019)
22. Chung, K.M., Fehr, S., Huang, Y.H., Liao, T.N.: On the compressed-oracle technique, and post-quantum security of proofs of sequential work. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 598–629. Springer (2021)
23. Cojocaru, A., Garay, J., Kiayias, A., Song, F., Wallden, P.: Quantum multi-solution bernoulli search with applications to bitcoin's post-quantum security. Quantum **7**, 944 (2023)
24. Ducas, L., Laarhoven, T., van Woerden, W.P.: The randomized slicer for cvpp: sharper, faster, smaller, batchier. In: IACR International Conference on Public-Key Cryptography. pp. 3–36. Springer (2020)
25. Durr, C., Hoyer, P.: A quantum algorithm for finding the minimum. arXiv preprint quant-ph/9607014 (1996)
26. Espitau, T., Joux, A., Kharchenko, N.: On a dual/hybrid approach to small secret lwe: A dual/enumeration technique for learning with errors and application to security estimates of fhe schemes. In: Progress in Cryptology–INDOCRYPT 2020: 21st International Conference on Cryptology in India, Bangalore, India, December 13–16, 2020, Proceedings 21. pp. 440–462. Springer (2020)

27. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z., et al.: Falcon: Fast-fourier lattice-based compact signatures over ntru. Submission to the NIST's post-quantum cryptography standardization process **36**(5), 1–75 (2018)
28. Giovannetti, V., Lloyd, S., Maccone, L.: Architectures for a quantum random access memory. Physical Review A—Atomic, Molecular, and Optical Physics **78**(5), 052310 (2008)
29. Giovannetti, V., Lloyd, S., Maccone, L.: Quantum random access memory. Phys. Rev. Lett. **100**, 160501 (Apr 2008). https://doi.org/10.1103/PhysRevLett.100.160501, https://link.aps.org/doi/10.1103/PhysRevLett.100.160501
30. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing. p. 212–219. STOC '96, Association for Computing Machinery, New York, NY, USA (1996). https://doi.org/10.1145/237814.237866, https://doi.org/10.1145/237814.237866
31. Guo, Q., Johansson, T.: Faster dual lattice attacks for solving lwe with applications to crystals. In: Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part IV 27. pp. 33–62. Springer (2021)
32. Hamoudi, Y., Liu, Q., Sinha, M.: The nisq complexity of collision finding. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 3–32. Springer (2024)
33. Heiser, M.: Improved quantum hypercone locality sensitive filtering in lattice sieving. Cryptology ePrint Archive, Paper 2021/1295 (2021), https://eprint.iacr.org/2021/1295, https://eprint.iacr.org/2021/1295
34. Hhan, M., Kim, J., Lee, C., Son, Y.: Let's meet ternary keys on babai's plane: A hybrid of lattice-reduction and meet-lwe. Cryptology ePrint Archive (2022)
35. Hhan, M., Yamakawa, T., Yun, A.: Quantum complexity for discrete logarithms and related problems. In: Annual International Cryptology Conference. pp. 3–36. Springer (2024)
36. Howgrave-Graham, N.: A hybrid lattice-reduction and meet-in-the-middle attack against ntru. In: Advances in Cryptology-CRYPTO 2007: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings 27. pp. 150–169. Springer (2007)
37. Hülsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P.: Ntru-hrss-kem. NIST submissions (2017)
38. Jaques, S., Rattew, A.G.: Qram: A survey and critique (2023)
39. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Quantum differential and linear cryptanalysis. IACR Transactions on Symmetric Cryptology **2016**(1), 71–94 (2016)
40. Kirshanova, E., Laarhoven, T.: Lower bounds on lattice sieving and information set decoding. In: Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II 41. pp. 791–820. Springer (2021)
41. Kirshanova, E., Mårtensson, E., Postlethwaite, E.W., Moulik, S.R.: Quantum algorithms for the approximate k-list problem and their application to lattice sieving. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019. pp. 521–551. Springer International Publishing, Cham (2019)

42. Laarhoven, T.: Search problems in cryptography: from fingerprinting to lattice sieving. Phd thesis 1 (research tu/e / graduation tu/e), Mathematics and Computer Science (Feb 2016), proefschrift
43. Laarhoven, T.: Approximate voronoi cells for lattices, revisited. Journal of Mathematical Cryptology **15**(1), 60–71 (2020)
44. MATZOV: Report on the Security of LWE: Improved Dual Lattice Attack (Apr 2022). https://doi.org/10.5281/zenodo.6412487, https://doi.org/10.5281/zenodo.6412487
45. Micciancio, D., Voulgaris, P.: Faster exponential time algorithms for the shortest vector problem. In: Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms. pp. 1468–1480. SIAM (2010)
46. Moiseev, E., Moiseev, S.: Time-bin quantum ram. Journal of Modern Optics **63**(20), 2081–2092 (2016)
47. Montanaro, A.: Quantum-walk speedup of backtracking algorithms. Theory OF Computing **14**(15), 1–24 (2018)
48. Nguyen, P.Q.: Boosting the hybrid attack on ntru: Torus lsh, permuted hnf and boxed sphere. In: NIST Third PQC Standardization Conference (2021)
49. Nguyen, P.Q., Vidick, T.: Sieve algorithms for the shortest vector problem are practical. Journal of Mathematical Cryptology **2**(2), 181–207 (2008). https://doi.org/doi:10.1515/JMC.2008.009, https://doi.org/10.1515/JMC.2008.009
50. Nielsen, M.A., Chuang, I.L.: Quantum computation and quantum information. Cambridge university press (2010)
51. NIST: Post-Quantum Cryptography Standardization. https://bit.ly/3lfzIub, accessed: 2024-05-14
52. Pohst, M.: On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. ACM Sigsam Bulletin **15**(1), 37–44 (1981)
53. Pouly, A., Shen, Y.: Provable dual attacks on learning with errors. In: Joye, M., Leander, G. (eds.) Advances in Cryptology – EUROCRYPT 2024. pp. 256–285. Springer Nature Switzerland, Cham (2024)
54. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
55. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
56. Schnorr, C.: A hierarchy of polynomial time lattice basis reduction algorithms. Theoretical Computer Science **53**(2), 201–224 (1987). https://doi.org/https://doi.org/10.1016/0304-3975(87)90064-8, https://www.sciencedirect.com/science/article/pii/0304397587900648
57. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review **41**(2), 303–332 (1999)
58. Wunderer, T.: A detailed analysis of the hybrid lattice-reduction and meet-in-the-middle attack. Journal of Mathematical Cryptology **13**(1), 1–26 (2019)
59. Zhandry, M.: How to record quantum queries, and applications to quantum indifferentiability. In: Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part II 39. pp. 239–268. Springer (2019)

A Omitted Proofs

Proof of Lemma 1. Write $S_i := [i \cdot S + 1, (i+1) \cdot S]$ for brevity. Write $X = \{x_1, ..., x_M\}$ (order can be arbitrary) and its subsets $X_i := \{x_{i \cdot S+1}, ..., x_{(i+1) \cdot S}\} = \{x_j : j \in S_i\} \subset X$ of size S for $i = 0, ..., \frac{M}{S} - 1$.

We define the algorithm A as follows: For each $0 \leq i \leq \frac{M}{S} - 1$, A stores X_i in QRAM. Define

$$Init_i : |0\rangle \mapsto \frac{\left(\sum_{j \in S_i} |j, x_j\rangle\right)}{\sqrt{S}}$$

using QRAM similarly to eq. (6). It applies quantum amplitude amplification to $O_P = I \otimes O_f$[16] and $Init_i$ for $O(\sqrt{S})$ times, and measures the result to obtain (j^*, x_{j^*}), and check if $f(x_{j^*}) = 1$. If true, it outputs x_{j^*} and halts. If there is no such x for all i, it returns $\perp$.

The running time of A requires at most $O\left(\frac{M}{S} \cdot \sqrt{S}\right) = O\left(\frac{T}{\sqrt{S}}\right)$ evaluations of f as we want, because we limit the number of iterations to $\sqrt{S}$ for each quantum amplitude amplification.

Next, we argue the correctness of the algorithm. Suppose that there exists $x^* \in X_i$ such that $f(x^*) = 1$. Then, Theorem 1 implies that the i-th quantum amplitude amplification can find $x \in X_i$ such that $f(x) = 1$ with a sufficiently high probability. □

Proof of Lemma 2. Write $S_i := [i \cdot S + 1, (i+1) \cdot S]$ for brevity. Let $X = \{x_1, \cdots, x_{M_1}\}, Y = \{y_1, \cdots, y_{M_2}\}$, and define subsets $X_i := \{x_k : k \in S_i\} \subseteq X$, $Y_j := \{y_\ell : \ell \in S_j\} \subset Y$ of size S for $i = 0, \cdots, \frac{M_1}{S} - 1$, $j = 0, \cdots, \frac{M_2}{S} - 1$.

With two QRAMs of size S, the algorithm A' runs amplitude amplification, similar to the proof in Lemma 1: After storing X_i and Y_j in each QRAM, define two operations;

$$Init_{i,j} := |0\rangle \mapsto \frac{\sum_{k \in S_i, \ell \in S_j} |k, \ell, x_k, y_\ell\rangle}{S} \tag{29}$$

with access to the QRAMs, and $O_P := I \otimes O_f$. As each X_i, Y_j are randomly selected, the expected number of solutions in $X_i \times Y_j$ is $E := \frac{K \cdot S^2}{M_1 \cdot M_2}$. The algorithm behaves differently, conditioned on the expected number of solutions.

1. If the expected number of solutions in the current QRAM is at least 1 (i.e., $KS^2 \geq M_1 M_2$), then we run the amplitude amplification multiple times to find most solutions. The number of calling $Init_{i,j}$ in Equation (29) and f is $E \cdot \sqrt{\frac{S^2}{E}} = S \cdot \sqrt{E} = \frac{\sqrt{K} \cdot S^2}{\sqrt{M_1 \cdot M_2}}$.
2. Otherwise, it needs $\sqrt{S^2}$ iterations to check if a solution exists in the current QRAM.

[16] Here, I acts on the index register.

There are $\frac{M_1 \cdot M_2}{S^2}$ pairs of (X_i, Y_j), therefore the total number of required iterations is $\sqrt{M_1 M_2 K}$ for the first case, and $\frac{M_1 M_2}{S}$ for the second case. For the correctness of A', Theorem 1 implies that the measurement output gives the solution pair $(x^*, y^*) \in X \times Y$ such that $f(x^*, y^*) = 1$ with a sufficiently high probability. □

A.1 The bounded QRAM search lower bound

In this section, we give the proof of Theorem 3. The proof uses the recording random functions [59,23]. The proof of the main lemma (Lemma 7) is inspired by the proofs in [32].

Preparation: Bernoulli random functions. The proof requires the recording technique for the Bernoulli random function introduced in [23]. We give a brief introduction below.

Let $|X| = 2^m$. We call $f : \{0,1\}^m \to \{0,1\}$ be a Bernoulli random function with parameter $0 < p < 1$ if $f(x) = 1$ holds with probability p independently. We denote the distribution of the Bernoulli random function by $B_{m,p}$, and α_f denotes the probability that the function f is sampled from $B_{m,p}$. It is well known that an algorithm having oracle access to a Bernoulli random function is identical to having oracle access to the purified Bernoulli random function that is defined by

$$\sum_{f \in B_{m,p}} \sqrt{\alpha_f}_F \, |f\rangle_F$$

where F is a $|X|$-qubit register. The standard query is computed by

$$\mathsf{StdBO} : |x, y\rangle \otimes \sqrt{\alpha_f}_F \, |f\rangle_F \mapsto (-1)^{y \cdot f(x)} \, |x, y\rangle \otimes \sqrt{\alpha_f}_F \, |f\rangle_F \tag{30}$$

We define the generalized Hadamard operation on a single qubit register

$$U_p : |b\rangle \mapsto \sqrt{1-p}\,|b\rangle + (-1)^b \sqrt{p}\,|b \oplus 1\rangle \text{ or } U_p = \begin{bmatrix} \sqrt{1-p} & -\sqrt{p} \\ \sqrt{p} & \sqrt{1-p} \end{bmatrix}.$$

Note that $\mathcal{U}_p := \otimes_{x \in \{0,1\}^m} U_p^x$ where U_p^x denotes U_p applied on the x-th qubit gives

$$\mathcal{U}_p \left| 0^{|X|} \right\rangle_F \mapsto \sum_{f \in B_{m,p}} \sqrt{\alpha_f}_F \, |f\rangle_F .$$

We define the dual query by

$$\mathsf{RBO} := (I \otimes \mathcal{U}_p^\dagger) \cdot \mathsf{StdBO} \cdot (I \otimes \mathcal{U}_p) \tag{31}$$

with the initial state $|0\rangle^{\otimes |X|}$. It is not hard to see that the output of any algorithm having access to StdBO (with the initial state $\sum_{f \in B_{m,p}} \sqrt{\alpha_f}_F \, |f\rangle_F$) and having access to RBO (with the initial state $(|0\rangle^{\otimes |X|})$ is identical.

We can compute the progress of the overall states under RBO using the following lemma, which is a Bernoulli analog of [22, Lemma 4.3]. The proof follows from the straightforward calculation.

Lemma 6. *The map* RBO *operates as follows, where* $|\cdot\rangle_x$ *denotes the* x*-th qutrit of* F*.*

$$\begin{aligned}
|x,+\rangle \otimes |b\rangle_x &\mapsto |x,+\rangle \otimes |b\rangle_x \text{ for arbitrary } b \\
|x,-\rangle \otimes |0\rangle_x &\mapsto |x,-\rangle \otimes ((1-2p)\,|0\rangle_x - 2\sqrt{p(1-p)}\,|1\rangle_x) \\
|x,-\rangle \otimes |1\rangle_x &\mapsto |x,-\rangle \otimes (-2\sqrt{p(1-p)}\,|0\rangle_x + (2p-1)\,|1\rangle_x)
\end{aligned}$$

Proof of the lower bound. Let A be an algorithm described in Theorem 3. Recall the oracle can be written as a Bernoulli random function $f : \{0,1\}^{|X|} \to \{0,1\}$ with parameter p, and S be the bound of the elements in the computational basis, and q be the number of queries. We consider the overall states of the algorithm and the oracle RBO initialized by

$$\left|\phi^{(0)}\right\rangle_{AF} |0\rangle_A \otimes \left|0^{|X|}\right\rangle_F.$$

Similarly, we write $\left|\phi^{(t)}\right\rangle_{AF}$ to denote the overall state after the t-th query. In the final step, we apply a projection $\Pi := \sum_x |x\rangle\langle x|_o \otimes |1\rangle\langle 1|_x$ where o denotes the output register of A, and $|1\rangle\langle 1|_x$ checks if the output is correct; the unspecified registers are not changed. For the final state $|\phi\rangle_{AF}$, the success probability is written by

$$p_{success} := \left\| \Pi (I \otimes \mathcal{U}) \left|\phi^{(q)}\right\rangle_{AF} \right\|^2. \tag{32}$$

By [23, Lemma 3.11], we have

$$\left\| \Pi (I \otimes \mathcal{U}) \left|\phi^{(q)}\right\rangle_{AF} \right\| \leq \sqrt{p} + \left\| \Pi \left|\phi^{(q)}\right\rangle_{AF} \right\|. \tag{33}$$

We will give the upper bound of the progress measure $p_t := \left\| \Pi \left|\phi^{(t)}\right\rangle_{AF} \right\|^2$ and relate it to the success probability. Using the following lemma and Equation (33), we have $\sqrt{p_{success}} \leq \sqrt{p_q} + \sqrt{p}$ which implies

$$p_{success} = O(\sqrt{S} \cdot pq)$$

as desired.

Lemma 7. *The following hold:*

1. $p_0 = 0$.
2. $p_{t+1} \leq p_t + c \cdot \sqrt{S} \cdot p$ *for some constant* $c > 0$.

Proof. The first item is obvious: Since there is no 1 in the F register in the initial state, $p_0 = 0$.

To prove the second item, we define the projectors $\Pi_Z^c := \otimes_{x \in Z} |0\rangle\langle 0|_x$, and

$$\Pi_Z := \sum_{x \in Z} |x\rangle\langle x|_o \otimes |1\rangle\langle 1|_x \otimes \Pi_{Z^c}^c, \Pi_{\neg Z} := \Pi - \Pi_Z, \text{ and } \Pi^c := I - \Pi$$

for a subset $Z \subset \{0,1\}^{|X|}$. Intuitively, Π_Z projects to the state where the entry 1 is found only in some $x \in Z$, and all other registers contain 0.

Applying a unitary on the register A does not change the progress measure. Thus, it suffices to prove the inequality for $\left|\phi^{(t+1)}\right\rangle_{AF} = \mathsf{RBO}\left|\phi^{(t)}\right\rangle_{AF}$ for any

$$\left|\phi^{(t)}\right\rangle_{AF} = \sum_{x,y \in X_t \times \{+,1\},D} \alpha_{xyD} \left|x,y\right\rangle \otimes \left|D\right\rangle_F \otimes \left|\phi_{xyD}\right\rangle$$

where $X_t \subset \{0,1\}^{|X|}$, which exists due to the assumption. Then, we have

$$p_{t+1} = \left\|\Pi \left|\phi^{(t+1)}\right\rangle\right\|^2 = \left\|\Pi \cdot \mathsf{RBO} \cdot (\Pi_{\neg X_t} + \Pi_{X_t} + \Pi^c)\left|\phi^{(t)}\right\rangle\right\|^2 \tag{34}$$

where we use $\Pi_{\neg X_t} + \Pi_{X_t} + \Pi^c = \Pi + \Pi^c = I$. This equals to

$$\left\|\Pi \cdot \mathsf{RBO} \cdot \Pi_{\neg X_t}\left|\phi^{(t)}\right\rangle\right\|^2 + \left\|\Pi \cdot \mathsf{RBO} \cdot (\Pi_{X_t} + \Pi^c)\left|\phi^{(t)}\right\rangle\right\|^2 \tag{35}$$

because the database of $\Pi \cdot \mathsf{RBO} \cdot \Pi_{\neg X_t}\left|\phi^{(t)}\right\rangle$ always contains $\left|1\right\rangle_x$ for some $x \notin X_t$, but the database of $\Pi \cdot \mathsf{RBO} \cdot (\Pi_{X_t} + \Pi^c)\left|\phi^{(t)}\right\rangle$ contains $\left|1\right\rangle_x$ only for $x \in X_t$. This is bounded above by

$$\leq \left\|\Pi_{\neg X_t}\left|\phi^{(t)}\right\rangle\right\|^2 + \left(\left\|\Pi_{X_t}\left|\phi^{(t)}\right\rangle\right\| + \left\|\Pi \cdot \mathsf{RBO} \cdot \Pi^c\left|\phi^{(t)}\right\rangle\right\|\right)^2 \tag{36}$$

$$= \left\|\Pi\left|\phi^{(t)}\right\rangle\right\|^2 + \left\|\Pi \cdot \mathsf{RBO} \cdot \Pi^c\left|\phi^{(t)}\right\rangle\right\|^2 \tag{37}$$

$$+ 2\left\|\Pi_{X_t}\left|\phi^{(t)}\right\rangle\right\| \cdot \left\|\Pi \cdot \mathsf{RBO} \cdot \Pi^c\left|\phi^{(t)}\right\rangle\right\|. \tag{38}$$

Using Lemma 6, it is easy to derive that $\|\Pi \cdot \mathsf{RBO} \cdot \Pi^c\left|\phi\right\rangle\|^2 \leq p$ for any $\left|\phi\right\rangle$.

By modifying the proof of [23, Lemma 3.11] (by just changing the role of the primal and dual domain and applying Π_{X_t}), we have

$$\left\|\Pi_{X_t}\left|\phi^{(t)}\right\rangle\right\| \leq \left\|\Pi_{X_t}(I \otimes \mathcal{U})\left|\phi^{(t)}\right\rangle\right\| + \sqrt{p}.$$

We have $\left\|\Pi_{X_t}(I \otimes \mathcal{U})\left|\phi^{(t)}\right\rangle\right\|^2$ is bounded by the probability that the algorithm finds a solution in X_t. This is again bounded by $\leq S \cdot p$ due to $|X_t| \leq S$, which is the probability that there exists a solution in X_t for the random Bernoulli function. This implies that $\left\|\Pi_{X_t}\left|\phi^{(t)}\right\rangle\right\| = O(\sqrt{S \cdot p})$.

Plugging this, we have the final upper bound

$$p_t + p + 2c\sqrt{S} \cdot p$$

for some constant $c > 0$. ($c = 10$ suffices.) This concludes the proof. □

B Proof of Bounded QRAM Lower Bounds

This section proves Theorem 11. To this end, we formally present the model of algorithms and costs, and the lower bounds of quantum sieving with a bounded QRAM.

B.1 Black-box near-neighbor algorithm and classical lower bound

In our model, a *black-box near-neighbor* algorithm A interacts with an oracle that keeps a list $L = (\boldsymbol{x}_1, ..., \boldsymbol{x}_n)$ of size $n = 2^{0.2075d+o(d)}$ and the sets A_i, B_i for $I \in [t]$ initialized by empty sets. The oracle decides random vectors $\boldsymbol{z}_1, ...\boldsymbol{z}_t$ from $\mathcal{S}^{d-1}$ uniformly at random and define Q_i and U_i by spherical caps as in Equation (24).

The algorithm A works over a working register $\mathbf{W}$, a query register $\mathbf{Q}$, and an arbitrarily long table register $\mathbf{T}$ and $\mathbf{X}$. The registers $\mathbf{W}$ and $\mathbf{Q}$ are initialized to $|0...0\rangle$ and A is allowed to apply an arbitrary unitary on $\mathbf{WQ}$ during its execution. The registers $\mathbf{X}$ and $\mathbf{T}$ are initialized by $|\boldsymbol{x}_1, ..., \boldsymbol{x}_n\rangle$ and $|0, 0, ...\rangle$. The algorithm A is allowed to make the following types of queries:

- *Insertion.* Measure the query register and interpret it as an element $(C, i, j) \in \{A, B\} \times [t] \times [n]$. The oracle updates $C_i \leftarrow C_i \cup \{\boldsymbol{x}_j\}$.
- *Sampling vectors from filters.* Apply the following operation on $\mathbf{Q}$ and $\mathbf{T}$ that works as follows in a computational basis:

$$|C, i, j, k\rangle_{\mathbf{Q}} |..., 0, ...\rangle_{\mathbf{T}} \mapsto |C, i, j, k\rangle_{\mathbf{Q}} |..., \boldsymbol{c}_j, ...\rangle_{\mathbf{T}} \tag{39}$$

  where $C \in \{A, B\}$, 0 is the k-th entry of $\mathbf{T}$ and $\boldsymbol{c}_j$ is the j-th element of C_i; if no such $\boldsymbol{c}_j$, define $\boldsymbol{c}_j = \mathbf{0}$.
- *Copying from* $\mathbf{X}$ *to* $\mathbf{T}$ Apply the following operation on $\mathbf{Q}, \mathbf{X}$ and $\mathbf{T}$:

$$|i, j\rangle_{\mathbf{Q}} |..., \boldsymbol{x}_i, ...\rangle_{\mathbf{X}} |..., 0, ...\rangle_{\mathbf{T}} \mapsto |i, j\rangle_{\mathbf{Q}} |..., \boldsymbol{x}_i, ...\rangle_{\mathbf{X}} |..., \boldsymbol{x}_i, ...\rangle_{\mathbf{T}} \tag{40}$$

  where 0 is the j-th entry of $\mathbf{T}$.
- *Sampling filters from vectors.* Apply the following operation on $\mathbf{Q}$ and $\mathbf{X}$ that works as follows in a computational basis:

$$|Z, i, 0, k\rangle_{\mathbf{Q}} |..., \boldsymbol{x}_k, ...\rangle_{\mathbf{X}} \mapsto |Z, i, \ell_i, k\rangle_{\mathbf{Q}} |..., \boldsymbol{x}_k, ...\rangle_{\mathbf{X}} \tag{41}$$

  where $Z \in \{C, Q\}$, $\boldsymbol{x}_k$ is the k-th entry of $\mathbf{X}$, $\{Z_{\ell_1}, ..., Z_{\ell_u}\}$ be the set of relevant (α- or β- depending on Z) filters such that $\ell_1 < ... < \ell_u$, and set $\ell_i = 0$ if $i > u$. If the third register of $\mathbf{Q}$ is non-zero, it does nothing.
- *Inner product.* Measure the second and third registers of $\mathbf{Q}$.[17] Apply the following operation on $\mathbf{Q}$ and $\mathbf{T}$ that works as follows in a computational basis:

$$|r, k, \ell\rangle_{\mathbf{Q}} |..., \boldsymbol{t}_k, ..., \boldsymbol{t}_\ell, ...\rangle_{\mathbf{T}} \mapsto |r \oplus b, k, \ell\rangle_{\mathbf{Q}} |..., \boldsymbol{t}_k, ..., \boldsymbol{t}_\ell, ...\rangle_{\mathbf{T}} \tag{42}$$

  where $\boldsymbol{t}_k, \boldsymbol{t}_\ell$ denote the k, ℓ-th entries of $\mathbf{T}$, and $b = 1$ if $\langle \boldsymbol{t}_k, \boldsymbol{t}_\ell \rangle \geq 1/2$, otherwise $b = 0$.

Except for the insertion, the inverse operations also can be queried. Note that each entry of the registers $\mathbf{T}, \mathbf{X}$ always contains 0 or a vector in L. Also note that the register $\mathbf{X}$ is always unchanged from the initial classical state.

[17] This measurement is to fix the indices of the inner product, which corresponds to the operation given in Cost Model 2. If we do not apply the measurement, it requires quantum RAM for quantum data.

If we consider the *classical* near-neighbor algorithms, $\mathbf{Q}$ is measured before applying the unitary. (We allow the other parts to be coherent.) The black-box algorithm cannot apply any other operation on $\mathbf{T}, \mathbf{X}$ besides the oracle queries, and comparison between two registers. This implies that the algorithm does not know anything about the input vectors except the queries: the indices of α- or β-relevant filters, the closeness of pairs,

We count the number of the last two types of queries (sampling filters and inner products) as the complexity measure. We also note that the black-box near-neighbor algorithm follows the algorithm outlined in Algorithm 2, especially Preprocess. We formally assume the following goal of the algorithm.

Definition 3. *After Preprocess, the hash-based near-neighbor algorithm is asked to find each tuple $(\boldsymbol{x}, \boldsymbol{y}) \in L \times L$ such that $(\boldsymbol{x}, \boldsymbol{y}) \in Q_i \times U_i$ holds for some i and $1 > \langle \boldsymbol{x}, \boldsymbol{y} \rangle \geq \cos\theta$ with probability at least 0.9. Alternatively, it needs to find at least $\Omega(n)$ such pairs with overwhelming probability.*

We assume $\theta = \pi/3$ and $t \geq \max(1/\mathcal{C}_d(\alpha), 1/\mathcal{C}_d(\beta), 1/\mathcal{W}_d(\alpha, \beta, \pi/3))$ to assure that for all near-neighbors have such an index with high probability.

Both `Query` and `FAS` methods indeed try to address the above task. The classical complexity lower bound can be derived from the following lemma. Recall that the classical algorithm always stores the queries and vectors classically.

Theorem 13. *The expected query complexities of the* classical *black-box near-neighbor algorithm with the purpose as in Definition 3 is at least*

$$\underbrace{nt \cdot \mathcal{C}_d(\beta)}_{\text{Preprocess}} + \underbrace{n \cdot \mathcal{C}_d(\alpha)/\mathcal{W}_d(\alpha, \beta, \pi/3)}_{\text{sampling filters wrt } Q_i} + \underbrace{n^2 \cdot \mathcal{C}_d(\alpha)\mathcal{C}_d(\beta)/\mathcal{W}_d(\alpha, \beta, \pi/3)}_{\text{inner products}} \tag{43}$$

up to a constant multiplicative factor, or at least $2^{0.2925d+o(d)}$. In particular, it must make $2^{0.2925d+o(d)}$ queries.

Proof. The formal proof should be involved with several probabilistic arguments, and intuitively, the proof shows that the queries in the main body are essential except for the case where the algorithm does not use the hash functions much. We give the proof sketch based on the average-case behavior.

The procedures for the first step are fixed, so the first term is obvious. Suppose that the number of inner products made by the algorithm is less than $n^{1.5} \geq 2^{0.2925d+o(d)}$; otherwise, it collapses to the "or" part at the end of the statement.

Note that if the algorithm knows $(\boldsymbol{x}, \boldsymbol{y}) \in Q_i \times U_i$, the probability that $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \geq 1/2$ is about $1/\mathcal{W}_d(\alpha, \beta, \pi/3)$ by Lemma 3. On the other hand, without such a constraint, $\boldsymbol{x}$ and $\boldsymbol{y}$ behave essentially randomly (because we assume that the vectors defining filters are uniformly distributed), the probability that $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \geq 1/2$ is about $1/(4/3)^{d/2+o(d)} = O(1/n)$ by Lemma 3. Therefore, the number of near-neighbors found by inner product queries without the knowledge of $(\boldsymbol{x}, \boldsymbol{y}) \in Q_i \times U_i$ is at most $n^{1.5} \cdot 1/n = n^{0.5}$ on average. This means that we need to find $n - n^{0.5} = \Omega(n)$ using the inner product queries with *the knowledge of the existence i*.

Fix $\boldsymbol{x} \in L$. For a near-neighbor $\boldsymbol{y} \in L$ of $\boldsymbol{x}$ (which exists with a high probability), the expected number of $i \in [t]$ satisfying the condition of Definition 3 is $t \cdot \mathcal{W}_d(\alpha, \beta, \pi/3)$. The algorithm must find one of such i using the sampling filter queries with a probability of at least 0.9. Since the vectors defining filters are random, the algorithm must make $t \cdot \mathcal{C}_d(\alpha)/(t \cdot \mathcal{W}_d(\alpha, \beta, \pi/3))$ sampling filter queries. The algorithm also requires making (almost) all inner product queries to find the near-neighbors, which is

$$n \cdot (\mathcal{C}_d(\alpha)/(\mathcal{W}_d(\alpha, \beta, \pi/3))) \cdot (n \cdot \mathcal{C}_d(\beta)) \tag{44}$$

which gives the lower bound we desired. Optimizing the complexity is essentially the same as in Theorem 4, which concludes the proof. □

B.2 Quantum lower bound

Now, we turn to the lower bound with a bounded QRAM setting. As outlined before, our strategy is, given a quantum black-box algorithm A, to construct a classical black-box simulation algorithm B that behaves almost like A. The result of the simulation can be described as follows.

Theorem 14. *Let A be a quantum black-box near-neighbor algorithm following Assumption 1 and Assumption 2. Suppose that a list of $n = 2^{0.2075d+o(d)}$ random vectors L in $\mathcal{S}^{d-1}$ is given as input, and that A makes at most q queries to the oracle. Then there exists another black-box near-neighbor algorithm B, given the same inputs, which makes at most $2^{2s} \cdot q$ classical queries to the oracle such that the output distribution of B and A are identical for any input.*

Since B always makes classical queries, the lower bound in Theorem 13 is applied to B. This implies that the query complexity q of A must satisfy $2^{2s} \cdot q \geq 2^{0.2925d+o(d)}$. Furthermore, if A solves the task given in Definition 3, B must solve the same task since the output distributions are identical. This concludes the proof of Theorem 11. We again note that the proof of this theorem is very similar to [35, Theorem 4.1].

Proof of Theorem 14. Let A be a quantum black-box near-neighbor algorithm following Assumption 1 and Assumption 2. Let $L = (\boldsymbol{x}_1, ..., \boldsymbol{x}_n)$. We construct a black-box algorithm B that simulates A as follows.

Initialization. For the simulation, the algorithm B maintains a "labeling function"

$$L : [n] \to [n] \cup \{\perp\}$$

by abusing the notation; we use $L(i)$ only for the labeling function to avoid confusion. Intuitively, $L(i) = \ell$ means, whenever A uses $\boldsymbol{x}_i$ (in the i-th entry of $\mathbf{X}$), the algorithm B uses ℓ instead of the vector $\boldsymbol{x}_i$ to construct their quantum

states.[18] In this way, B will always know its exact state at any time of the execution.

All labels are initially set by $\perp$, which means they are not specified yet. The algorithm B initializes the function L as follows: For $i = 1, ..., n$, set $L(\boldsymbol{x}_i)$ uniformly a random element from $[M]$ conditioned that it is not used before. B creates the following state

$$|0, ..., 0\rangle_{\mathbf{WQ}} \otimes |\boldsymbol{x}_1, ..., \boldsymbol{x}_n\rangle_{\mathbf{X}} \otimes |\perp, ..., \perp\rangle_{\mathbf{T}'}$$

which is identical to the initial state of A except for the zero vectors in $\mathbf{T}$ is replaced by $\perp$. Whenever A has the state

$$\sum_{wq, T=(\boldsymbol{t}_1, \boldsymbol{t}_2, ...)} \alpha_{wq,T} \, |wq\rangle_{\mathbf{WQ}} \otimes |\boldsymbol{x}_1, ..., \boldsymbol{x}_n\rangle_{\mathbf{X}} \otimes |\boldsymbol{t}_1, \boldsymbol{t}_2, ...\rangle_{\mathbf{T}}$$

such that $\boldsymbol{t}_i = \boldsymbol{x}_{j_i}$ for $i = 1, 2, ..$, B will construct the state

$$\sum_{wq, J_T=(j_1, j_2, ...)} \alpha_{wq,J_T} \, |wq\rangle_{\mathbf{WQ}} \otimes |\boldsymbol{x}_1, ..., \boldsymbol{x}_n\rangle_{\mathbf{X}} \otimes |L(j_1), L(j_2), ...\rangle_{\mathbf{T}'} .$$

Local operation. When A applies some operation U on its local registers $\mathbf{WQ}$, B also applies the same operation U on its $\mathbf{WQ}$.

Insertion. When A applies the insertion query, B does the same query. The definition of the insertion query forces to measure the query register, B can apply the same insertion operation made by A.

Sampling vectors from filters. Suppose A makes the sampling vectors from filters query with the input state

$$\sum_{(C,i,j) \in J, k} \alpha_{C,i,j,k} \, |C, i, j, k\rangle_{\mathbf{Q}} \otimes |..., 0, ...\rangle_{\mathbf{T}}$$

for some set J. For convenience, we assume that k is fixed; the general case can be dealt with analogously. By Assumption 1, $|J| \leq 2^s$ holds.

B makes the classical sampling vectors from filter queries for all (C, i, j, k) for $(C, i, j) \in J$[19] and let $\boldsymbol{t}_{C,i,j}$ be the sampled vector stored in the k-th entry of $\mathbf{T}$, finds $i_{C,i,j} \in [n]$ such that $\boldsymbol{t}_{C,i,j} = \boldsymbol{x}_{i_{C,i,j}}$, and applies the inverse operation of the previous query to remove the k-th entry vector from $\mathbf{T}$. Then, B computes

$$\sum_{(C,i,j) \in J, k} \alpha_{C,i,j,k} \, |C, i, j, k\rangle_{\mathbf{Q}} \otimes |..., L(i_{C,i,j}), ...\rangle_{\mathbf{T}'}$$

from the current state $\sum_{(C,i,j) \in J, k} \alpha_{C,i,j,k} \, |C, i, j, k\rangle_{\mathbf{Q}} \otimes |..., 0, ...\rangle_{\mathbf{T}'}$.

[18] The use of the labeling function is because the algorithm B does not have access to the register $\mathbf{X}$ and $\mathbf{T}$ without queries, so it cannot construct the coherent states over the input vectors only using the classical queries; instead, B constructs the coherent states over the labels.

[19] B knows the whole state perfectly, so it can be recovered.

Copying from $\mathbf{X}$ *to* $\mathbf{T}$. B applies the same operation as A.

Sampling filters from vectors. This is almost the same as the sampling vectors. This time, the algorithm's state is

$$\sum \alpha_{Z,i,k} \left|Z,i,j,k\right\rangle_{\mathbf{Q}} \mapsto \sum \alpha_{Z,i,k} \left|Z,i,\ell_i,k\right\rangle_{\mathbf{Q}}$$

for A, where $Z \in \{C,Q\}$, $\{Z_{\ell_1},...,Z_{\ell_u}\}$ be the set of relevant filters such that $\ell_1 < ... < \ell_u$. We omit the register $\mathbf{X}$ because it is fixed. Here, the number of (Z,i,k) such that $\alpha_{Z,i,k} \neq 0$ must be at most 2^s because of Assumption 2. B collects such (Z,i,k) and makes the classical sampling filter queries for all (Z,i,k) and obtain ℓ_i (wrt Z,k) and construct the corresponding state. It makes at most 2^s *classical* sampling filters from vectors queries for each quantum query by A.

Inner product. When A applies the inner product query on

$$\sum_{r,T} \alpha_{r,T} \left|r,k,\ell\right\rangle_{\mathbf{Q}} \left|...,\boldsymbol{t}_k,...,\boldsymbol{t}_\ell,...\right\rangle_{\mathbf{T}} \mapsto \sum_{r,T} \alpha_{r,T} \left|r \oplus b,k,\ell\right\rangle_{\mathbf{Q}} \left|...,\boldsymbol{t}_k,...,\boldsymbol{t}_\ell,...\right\rangle_{\mathbf{T}},$$

B retrieves the information in the k-th and ℓ-th registers of $\mathbf{T}'$. Note that the sampling vector queries are the only way to make the coherent state over $\mathbf{T}$, and by Assumption 1, each entry of $\mathbf{T}$ has at most 2^s vectors with nonzero amplitudes. Thus, we can write $\boldsymbol{t}_{k,1},...\boldsymbol{t}_{k,2^s},\boldsymbol{t}_{\ell,1},...,\boldsymbol{t}_{\ell,2^s}$ be such vectors with nonzero amplitudes. B computes the indices $i_{k,1},...,i_{k,2^s},i_{\ell,1},...,i_{\ell,2^s}$ such that $\boldsymbol{x}_{i_{k,\star}} = \boldsymbol{t}_{k,\star}$ and the same for ℓ. Using the classical inner product queries, B compute $1_{\langle \boldsymbol{x}_{i_{k,a}},\boldsymbol{x}_{i_{\ell,b}}\rangle \geq 1/2} = 1_{\langle \boldsymbol{t}_{k,a},\boldsymbol{t}_{k,b}\rangle \geq 1/2}$ for all $a,b \in [2^s]$. From this, B can compute the same query using 2^{2s} classical inner product queries.

Finalization. When A outputs a near-neighbor pair $(\boldsymbol{x}_i,\boldsymbol{x}_j)$, it must contain (i,j) in its local register $\mathbf{W}$. B does the same and outputs (i,j).

As observed above, the truncated states in register $\mathbf{WQ}$ of A and B are identical, thus the output distributions of A and B are identical for any input. The complexity of B is at most 2^{2s} times the query complexity of A, because each quantum query can be simulated by at most 2^{2s} classical query. This concludes the proof. □

Table of Contents