



*entropy*



Article

---

# Phase Coordinate Uncomputation in Quantum Recursive Fourier Sampling

---

Christoffer Hindlycke, Niklas Johansson and Jan-Åke Larsson

Topic

Quantum Information and Quantum Computing, 2nd Volume

Edited by

Dr. Durdu Guney and Dr. David Petrosyan



<https://doi.org/10.3390/e27060596>

Article

# Phase Coordinate Uncomputation in Quantum Recursive Fourier Sampling

Christoffer Hindlycke <sup>\*,†</sup> , Niklas Johansson  and Jan-Åke Larsson <sup>†</sup> 

Department of Electrical Engineering, Linköping University, 581 83 Linköping, Sweden; niklas.johansson@liu.se (N.J.); jan-ake.larsson@liu.se (J.-Å.L.)

\* Correspondence: christoffer.hindlycke@liu.se

† These authors contributed equally to this work.

**Abstract:** Recursive Fourier Sampling (RFS) was one of the earliest problems to demonstrate a quantum advantage, and is known to lie outside the Merlin–Arthur complexity class. This work contains a new description of quantum algorithms in phase space terminology, demonstrating its use in RFS, and how and why this gives a better understanding of the quantum advantage in RFS. Most importantly, describing the computational process of quantum computation in phase space terminology gives a much better understanding of why uncomputation is necessary when solving RFS: the advantage is present only when phase coordinate garbage is uncomputed. This is the underlying reason for the limitations of the quantum advantage.

**Keywords:** quantum algorithms; phase kickback; quantum advantage; uncomputation; quantum oracles

## 1. Introduction

Recursive Fourier Sampling (RFS) [1,2], originally introduced by Bernstein and Vazirani in 1993, was one of the earliest computational problems for which there existed a quantum algorithm apparently demonstrating an advantage over any classical algorithm. It is not known which complexity class RFS belongs to, but in the oracle paradigm it has been shown to lie outside Merlin–Arthur (MA) [3], which contains the Non-Deterministic Polynomial Time (NP) complexity class. Some further conjectures on which complexity class RFS may belong to are made in Ref. [4]. In spite of this, RFS has received scant attention, possibly in part owing to the implicit way in which it is formulated.

In this work, we provide a small generalization of RFS using updated notation, connect our generalization and previous formulations, and prove that uncomputation is necessary in a quantum solution to each RFS subproblem; this was already implied by existing complexity bounds. Our formulation does more since it enables a direct comparison with classical reversible computing where (computational coordinate) garbage is uncomputed to enable reversibility. In RFS, the uncomputation of phase coordinate garbage enables the quantum advantage. The mechanism of phase kickback [5–7] is an important quantum computational property that enables the quantum solution of RFS, in line with previous results [7] on, for instance, the Bernstein–Vazirani algorithm (which is equal to RFS for recursion depth 1). Here, we extend this picture to a full phase space description.

This is done by reformulating the oracles employed by RFS into the terminology of *conjugate pair oracles* that we define in this work; a conjugate pair oracle outputs a *conjugate pair* of two phase space coordinates, one in the computational basis and one in the phase basis. Describing and thereby understanding quantum computation in terms of phase



check for updates

Academic Editors: Durdu Guney and David Petrosyan

Received: 14 April 2025

Revised: 29 May 2025

Accepted: 1 June 2025

Published: 2 June 2025

**Citation:** Hindlycke, C.; Johansson, N.; Larsson, J.-Å. Phase Coordinate Uncomputation in Quantum Recursive Fourier Sampling. *Entropy* **2025**, *27*, 596. <https://doi.org/10.3390/e27060596>

**Copyright:** © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

space coordinates [8] has previously yielded a number of interesting results, such as the stabilizer formalism [9,10], several simple toy models, which nonetheless demonstrate a surprising number of quantum behaviors [7,11], and above all, quantum error-correcting codes [12–14]. This means of description thus allows for a rich representation while remaining accessible, in the sense that we may more easily understand what is happening during our computations. As RFS lies outside **MA**, our results lend credence to the conjecture [7] that oracle-paradigm bounds for the quantum advantage may be imposed by limited communication capacity rather than limited computational capacity: Any advantage seems to stem from the quantum oracle outputting (communicating) additional data compared to a classical oracle.

Below, we assume familiarity with basic linear algebra and quantum computation; for a comprehensive introduction, see [6]. We denote by  $x_k$  a bitstring of some fixed length  $n_k$ , and use “ $\cdot$ ” to denote the dot product modulo 2. The rest of this work is organized as follows. In Section 2, we describe Fourier Sampling, how to perform it using classical and quantum machines and how it can be used to solve a certain decision problem; we also explain the quantum algorithm’s use of phase kickback. Here, we make use of standard notation. Next, in Section 3, we describe Recursive Fourier Sampling, and how to perform it using classical and quantum machines. The presentation is more streamlined than in earlier works: A comparison to earlier presentations can be found in Appendix A. Then, in Section 4, we go on to describe the contributions of the present work by introducing the notion of conjugate pair oracles, use this notion to show that uncomputing phase coordinate garbage is the enabling mechanism behind the quantum solution of RFS, and provide a proof of why uncomputation cannot be avoided. We conclude by summarizing our main findings, briefly discuss the complexity of RFS, and point out some future avenues of research.

## 2. Fourier Sampling and Phase Kickback

Given a function  $\varphi$  encoded into the coefficients of the quantum state  $|\varphi\rangle = \sum_{x=0}^{N-1} \varphi(x) |x\rangle$ , the Quantum Fourier Transform (QFT) gives

$$QFT |\varphi\rangle = \sum_{\chi=0}^{N-1} \widehat{\varphi}(\chi) |\chi\rangle. \quad (1)$$

A computational basis measurement outcome  $X$  will then be a sample from the distribution  $P(X = \chi) = |\widehat{\varphi}(\chi)|^2$ , known as Fourier Sampling, from the power spectrum of the function  $\varphi$ . Samples essentially give the inverse image of the power spectrum, e.g., frequencies absent from  $\widehat{\varphi}$  will occur with probability 0.

The bit-wise Quantum Fourier Transform mod 2 (the Hadamard transform) was used by Bernstein and Vazirani in 1993 [1] to solve the following problem: given an oracle for the Boolean function  $f$  promised to be linear, i.e.,

$$f(x) = s \cdot x, \quad (2)$$

determine the unknown bitstring  $s$ . The linear choice of  $f$  is part of the original problem description. There is a version utilizing a non-linear (quadratic)  $f$  [15] but there exists no recursive variant of this problem.

A classical algorithm would need to call the corresponding classical reversible oracle

$$O_f(x, y) = (x, y + f(x)) \quad (3)$$

a total of  $n$  times (see Algorithm 1), since it can extract at most one bit of information from each call, available in the target bit  $y$ . Querying  $f$  with the argument  $x = 1_j$  where  $1_j$  is the bitstring with only the  $j$ th bit set, the function output is  $(s)_j$ , the  $j$ th bit of the bitstring  $s$ .

---

**Algorithm 1** Classical Fourier Sampling

---

**Classical oracle:**  $O_f$   
**Control:** none  
**Bias/Target:** bitstring  $y$  of length  $n$   
1: **for**  $j := 1, \dots, n$  **do**  
2:     **create ancilla** bit string  $c = 1_j$   
3:     **apply**  $O_f$  to  $(c, (y)_j)$   
4:     **discard ancilla**  $c$

---

The promise (2) tells us that if the initial bias  $y = 0$ , the output target  $y$  will have the value  $s$ , since  $f(1_j) = s \cdot 1_j = (s)_j$ .

An appropriate function for Fourier Sampling is  $(-1)^{f(x)}$  because the Hadamard transform of  $(-1)^{f(x)} = 1 - 2f(x)$  is  $1 - 2\hat{f}(\chi) = \delta_{\chi s}$ , the Kronecker delta function, which equals 1 if  $\chi = s$  and 0 otherwise. If one can generate a quantum state containing  $(-1)^{f(x)}$  in its coefficients, Fourier Sampling will give  $s$  (with probability 1). Quantum algorithms provide a way to accomplish this through a procedure known as phase kickback [5,7].

Phase kickback uses a quantum function oracle  $U_f$  that preserves phase information,

$$U_f \sum_{x,y} c_{x,y} |x, y\rangle = \sum_{x,y} c_{x,y} |x, y + f(x)\rangle, \tag{4}$$

and elements of the phase basis in control and target registers, to move the function output from the target computational basis state in the right-hand side above, to the phase of the coefficients in the controlling register. Applying  $U_f$  to the phase basis elements gives

$$\begin{aligned} U_f H^{\otimes n+1} |\chi, v\rangle &= U_f \sum_{x,y} (-1)^{\chi \cdot x} |x\rangle (-1)^{vy} |y\rangle = \sum_{x,y} (-1)^{\chi \cdot x} |x\rangle (-1)^{vy} |y + f(x)\rangle \\ &= \sum_{x,y} (-1)^{\chi \cdot x} |x\rangle (-1)^{v(y-f(x))} |y\rangle = \sum_{x,y} (-1)^{\chi \cdot x - v f(x)} |x\rangle (-1)^{vy} |y\rangle. \end{aligned} \tag{5}$$

For Quantum Fourier Sampling, we use the Hadamard transform and bitwise addition modulo 2, and the Fourier variables are bitstrings  $x, \chi$  of length  $n$  and bits  $y, v$ . Then, the Fourier Sampling promise of Equation (2) gives

$$H^{\otimes n+1} U_f H^{\otimes n+1} |\chi, v\rangle = H^{\otimes n+1} \sum_{x,y} (-1)^{(\chi+vs) \cdot x} |x\rangle (-1)^{vy} |y\rangle = |\chi + vs, v\rangle. \tag{6}$$

Bernstein and Vazirani [1] write this as an algorithm that solves the Fourier Sampling problem in one call to the quantum oracle, see Algorithm 2.

---

**Algorithm 2** Quantum Fourier Sampling

---

**Quantum oracle:**  $U_f$   
**Control:** none  
**Bias/Target:** qubit string  $|\psi_x\rangle$  of length  $n$   
1: **create ancilla** qubit  $|\psi_y\rangle := |1\rangle$   
2: **apply**  $H^{\otimes n+1}$  to  $|\psi_x, \psi_y\rangle$   
3: **apply**  $U_f$  to  $|\psi_x, \psi_y\rangle$   
4: **apply**  $H^{\otimes n+1}$  to  $|\psi_x, \psi_y\rangle$   
5: **discard ancilla**  $|\psi_y\rangle$

---

The promise (6) tells us that if the initial bias  $|\psi_x\rangle = |\chi\rangle = |0\rangle$ , measurement of the output  $|\psi_x\rangle$  will give the outcome  $s$  (with probability 1), and also that the ancilla is fixed to a known value in step 5 so can be safely discarded.

The problem as stated so far is a sampling problem: sample from a specific probability distribution (output  $s$  with probability 1). In what follows, it will be useful to re-state the problem as a decision problem, that determines if a given bitstring  $x_1$  belongs to the language  $\mathcal{L}_{f_1}$ . In the decision problem version of Fourier Sampling, we are given a Boolean function of two arguments, promised to obey

$$f_2(x_1, x_2) = s_1(x_1) \cdot x_2, \tag{7}$$

and another Boolean function  $g_1(x_1, s_1(x_1))$  that gives the answer to the problem upon input of  $s_1(x_1)$  (this latter function can be referred to as  $f_1(x_1)$ ). Given oracle access to  $f_2$  and  $g_1$ , we need to use classical or Quantum Fourier Sampling on the last argument of  $f_2$  to obtain  $s_1(x_1)$  so that we can use  $g_1$  to determine if  $x_1$  belongs to  $\mathcal{L}_{f_1}$  or not, to solve the decision problem.

Note that if  $g_1$  is linear in the second argument and independent of the first, then there exists an  $x^*$  so that  $g_1(x_1, x_2) = x^* \cdot x_2$ , and  $f_1(x_1) = g_1(x_1, s_1(x_1)) = x^* \cdot s_1(x_1) = f_2(x_1, x^*)$ . Such a decision problem can be solved in a single call to  $f_2$ ; thus problems with a linear  $g_1$  (sometimes known as a parity function) are less interesting to study [1].

### 3. Recursive Fourier Sampling

There are, to the best of our knowledge, three formulations of RFS. The original was introduced by Bernstein and Vazirani in 1993 [1], further expanded on by the same authors in 1997 [2], and presented in a slightly different manner by Vazirani in 2002 [3]. In 2003, Aaronson [16] re-formulated RFS somewhat, in part by removing the bitstring size limitations. Finally, in 2008 Johnson [4] essentially combined elements of the two versions in another formulation. We shall make use of the definitions of Ref. [3], but remove the bitstring size limitations in keeping with [16], and add subscripts to function sequences for indexing the recursion level. A comparison of the notation for the different versions can be found in Appendix A.

RFS is a decision problem: it uses a so-called Fourier Sampling tree  $\{f_k : 1 \leq k \leq l + 1\}$  to decide whether a bitstring  $x_1$  belongs to a language  $\mathcal{L}_{f_1}$  or not. A Fourier Sampling tree is a Boolean function sequence such that for  $k > 1$

$$f_k(x_1, \dots, x_k) = s_{k-1}(x_1, \dots, x_{k-1}) \cdot x_k. \tag{8}$$

We say that the sequence  $f_k$  is *derived from* the sequence  $g_k$  for  $1 \leq k \leq l$  (and  $g_k$  *specifies* the Fourier Sampling tree  $f_k$ ) if

$$f_{k-1}(x_1, \dots, x_{k-1}) = g_{k-1}(x_1, \dots, x_{k-1}, s_{k-1}(x_1, \dots, x_{k-1})). \tag{9}$$

Now, given oracles for  $g_k$ ,  $1 \leq k \leq l$ , and  $f_{l+1}$ , the challenge is to determine  $f_1(x_1)$ , which in turn, tells us if  $x_1$  is in the language  $\mathcal{L}_{f_1}$  or not. Note that oracle access to  $f_{l+1}$ , though not explicitly stated in [3], is necessary to perform Quantum Fourier Sampling as described there.

A classical algorithm would need to solve this recursively, at each step using the corresponding classical reversible oracle

$$O_{f_k}(x_1, \dots, x_k, y) = (x_1, \dots, x_k, y + f(x)) \tag{10}$$

a total of  $n_k$  times, since it can only extract (at most) one bit of information from each call, available in the target bit  $y$ . We here rewrite the explicit classical algorithm by McKague [17] in our notation, see Algorithm 3.

**Algorithm 3** Classical Recursive Fourier Sampling

**Level:**  $1 \leq k \leq l$   
**Classical oracles:**  $O_{f_{k+1}}$  and  $O_{g_k}$   
**Control:** bitstrings  $x_j, 1 \leq j \leq k$   
**Bias/Target:** single bit  $y$

- 1: **create ancilla** bitstring  $x_{k+1} = 0$
- 2: **for**  $j = 1, \dots, n_{k+1}$  **do**
- 3:     **create ancilla** bit string  $c = 1_j$  (of length  $n_{k+1}$ )
- 4:     **apply**  $O_{f_{k+1}}$  to  $(x_1, \dots, x_k, c, (x_{k+1})_j)$
- 5:     **discard ancilla**  $c (= 1_j)$
- 6: **apply**  $O_{g_k}$  to  $(x_1, \dots, x_{k+1}, y)$
- 7: **discard ancilla**  $x_{k+1}$

From the promise (2), it follows that oracle access to  $f_{k+1}$  gives us  $f(x_1, \dots, x_k, 1_j) = (s_k(x_1, \dots, x_k))_j$ , so that  $x_{k+1} = s_k(x_1, \dots, x_k)$  after the recursion for-loop. It then follows from the promise (9) that oracle access to  $g_k$  with this value of  $x_{k+1}$  gives oracle access to  $f_k$ . That this holds for all  $k$  follows by induction, in particular, we obtain oracle access to  $f_1$  so that we can calculate  $f_1(x_1)$  [17].

This algorithm needs  $n_2 n_3 \dots n_{l+1}$  calls to  $f_{l+1}$  and  $n_2 n_3 \dots n_k$  calls to  $g_k, 1 < k \leq l$ , so the algorithm has the complexity  $O(\prod_{k=2}^{l+1} n_k)$ , which in the case of equal lengths  $n_k = n$  is  $O(n^l)$  [16]; the corresponding lower bound is then  $\Omega(n^l)$  [4], meaning we have a strict bound  $\Theta(n^l)$ . Calculating an exact bound for a problem given string lengths  $n_1, \dots, n_k$  is easily done via the formula above.

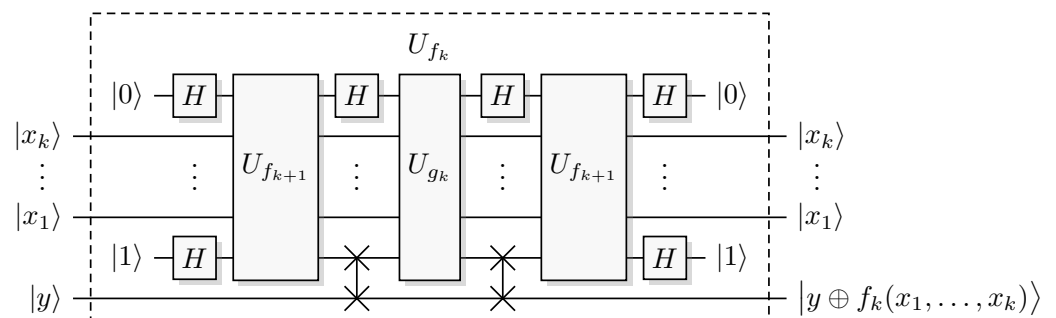
A quantum oracle would, also here, be assumed to preserve phase information

$$U_{f_k} \sum_{x_1, \dots, x_k, y} c_{x_1, \dots, x_k, y} |x_1, \dots, x_k, y\rangle = \sum_{x_1, \dots, x_k, y} c_{x_1, \dots, x_k, y} |x_1, \dots, x_k, y + f(x_1, \dots, x_k)\rangle. \tag{11}$$

Let  $H_j$  denote the unitary that applies Hadamards on the qubits of  $|x_j\rangle$ . Then, the Fourier Sampling promise of Equation (8), with parameters  $x_1, \dots, x_{k-1}$ , gives

$$(H_k \otimes H) U_{f_k} (H_k \otimes H) |x_1, \dots, x_{k-1}, \chi, v\rangle = |x_1, \dots, x_{k-1}, \chi + v s_{k-1}(x_1, \dots, x_{k-1}), v\rangle. \tag{12}$$

The assumption (11) implies phase is preserved for a superposition of such input states. A quantum algorithm can now use such a quantum oracle and a different recursion [17], see Figure 1 and Algorithm 4.



**Figure 1.** Quantum Recursive Fourier Sampling. The focus of the uncomputation is to reset the ancillary systems  $|x_{k+1}\rangle$  to  $|0\rangle$  and  $|y'\rangle$  to  $|1\rangle$ . The focus changes and becomes much clearer in the conjugate pair oracle paradigm in Section 4.

---

**Algorithm 4** Quantum Recursive Fourier Sampling

---

**Level:**  $1 \leq k \leq l$

**Quantum oracles:**  $U_{f_{k+1}}$  and  $U_{g_k}$

**Controls:** qubit strings  $|\psi_{x_j}\rangle, 1 \leq j \leq k$

**Bias/Target:** single qubit  $|\psi_y\rangle$

- 1: **create ancilla** qubit string  $|\psi_{x_{k+1}}\rangle := |0\rangle$  and qubit  $|\psi_{y'}\rangle := |1\rangle$
  - 2: **apply**  $H^{\otimes n_{k+1}+1}$  to  $|\psi_{x_{k+1}}, \psi_{y'}\rangle$
  - 3: **apply**  $U_{f_{k+1}}$  to  $|\psi_{x_1}, \dots, \psi_{x_{k+1}}, \psi_{y'}\rangle$
  - 4: **apply**  $H^{\otimes n_{k+1}}$  to  $|\psi_{x_{k+1}}\rangle$
  - 5: **apply**  $U_{g_k}$  to  $|\psi_{x_1}, \dots, \psi_{x_{k+1}}, \psi_{y'}\rangle$
  - 6: **apply**  $H^{\otimes n_{k+1}}$  to  $|\psi_{x_{k+1}}\rangle$
  - 7: **apply**  $U_{f_{k+1}}$  to  $|\psi_{x_1}, \dots, \psi_{x_{k+1}}, \psi_{y'}\rangle$
  - 8: **apply**  $H^{\otimes n_{k+1}+1}$  to  $|\psi_{x_{k+1}}, \psi_{y'}\rangle$
  - 9: **discard ancillas**  $|\psi_{x_{k+1}}, \psi_{y'}\rangle$
- 

With the ancillas  $|\psi_{x_{k+1}}, \psi_{y'}\rangle = |\chi, v\rangle = |0, 1\rangle$ , it follows from Equation (12) that  $|\psi_{x_{k+1}}\rangle = |s_k(x_1, \dots, x_k)\rangle$  after step 4. It then follows from the promise (9) that oracle access to  $g_k$  through  $U_{g_k}$  with this state  $|\psi_{x_{k+1}}\rangle$  gives oracle access to  $f_k$ , creating  $U_{f_k}$ . Then steps 6–8 reset  $|\psi_{x_{k+1}}\rangle = |0\rangle$  so that it can be safely discarded, usually motivated by the need to enable interference [2,16]; the actual underlying reason is to ensure that phase preservation of  $U_{g_k}$  gives phase preservation of  $U_{f_k}$ , we will return to this point below. That we have oracle access to  $f_k$  for all  $k$  follows by induction, in particular, we obtain oracle access to  $f_1$  so that we can calculate  $f_1(x_1)$  [17]. c This algorithm needs  $2^l$  calls to  $U_{f_{l+1}}$ , so the algorithm has the complexity  $O(2^l)$ , and the corresponding lower bound is  $\Omega(2^l)$  [4,16], meaning we have a strict bound  $\Theta(2^l)$ .

#### 4. Conjugate Pair Oracles and Phase Coordinate Uncomputation

We are now almost in a position to define and use conjugate pair oracles. This will yield an alternative description of RFS, in turn, enabling a proof that uncomputation is necessary in RFS, which (together with the conjugate pair formalism) is the main contribution of this work. As a conjugate pair oracle acts on a conjugate pair, we begin by defining the latter, as a representation based on phase space coordinates [8] and this idea allows for our results.

The Hadamard arrangement of the presented quantum algorithms is usually only seen as a convenient way to move  $f$  from the target register into the phase of the coefficients of the control register, to enable Fourier Sampling from that register [1]. The  $|\psi_y\rangle = |v\rangle$  bias input is converted into a negative phase  $(-1)^v$ , which then enables the “phase kickback” of the function output into the phase of the control system, to enable the  $|\psi_y\rangle = |s\rangle$  output if  $v = 1$ . However, this can be viewed in a different manner. The Fourier coordinate is a canonically conjugate phase space coordinate, often referred to as “the phase basis”.

In textbooks on quantum mechanics, one typically first encounters the canonically conjugate pair  $[x, p]$  of position  $x$  and momentum  $p$ . The brackets are here meant to indicate “the canonically conjugate pair” of physical quantities, not the commutator between two Hermitian operators as is commonplace in quantum mechanics. In quantum computing, the appropriate conjugate pair is computational basis and phase basis, the pair  $\mathcal{X} = [x, \chi]$  of computational basis bitstring  $x$  and phase bitstring  $\chi$ , of equal length. It is this which we take as our definition of a conjugate pair, two bitstrings of equal length; this is then the length of the conjugate pair.

In quantum mechanics, if the state of the studied system is an element of the computational basis, then  $x$  is well-defined but  $\chi$  is not; if the state is an element of the phase basis, then  $\chi$  is well-defined but  $x$  is not. This is a consequence of the uncertainty rela-

tion [7]. Even partial knowledge of  $\chi$  forces the value of  $x$  to be partially unknown, or really, partially unknowable, and vice versa. Therefore, we cannot access both entries of the conjugate pair  $\mathcal{X} = [x, \chi]$ , we must choose one of them. In other words: for a conjugate pair (when used in quantum computation) we may have that the computational basis bitstring is well-defined, or that the phase bitstring is well-defined, or that neither is well-defined (but never that both are well-defined).

Creating a computational basis state can now be seen as writing a well-defined value  $x$  into the computational basis part of the conjugate pair  $\mathcal{X} = [x, \chi]$ , and measuring in the computational basis can be seen as reading off the  $x$  value. Likewise, creating a phase basis state can now be seen as writing a well-defined value  $\chi$  into the phase basis part of the conjugate pair  $\mathcal{X} = [x, \chi]$ , and measuring in the phase basis can be seen as reading off the  $\chi$  value.

The Hadamard operation or bitwise Fourier transform mod 2 moves a potential well-defined value from the computational basis part to the phase basis part of the state and vice versa; an appropriate notation for a system where  $\mathcal{X}$  has length  $n$  would be

$$\mathcal{H}^n \mathcal{X} = \mathcal{H}^n [x, \chi] = [\chi, x]. \tag{13}$$

Re-writing Quantum Fourier Sampling from this point of view, the first two steps of the algorithm write information into the phase basis, and the final two steps read information from the phase basis [7]. The promise associated with a quantum oracle, that phase is preserved among the components of a superposition, now implies a simpler promise: that the oracle can calculate not one but two kinds of output. The standard function map

$$x \mapsto f(x), \tag{14}$$

is still available, with the standard control and bias/target registers. However, a second function map is also available, the phase kickback map that we will denote

$$v \mapsto \check{f}(v), \tag{15}$$

from the phase coordinate of the target/bias register to the phase coordinate of the control. The behavior of these two function maps can now be collected into a joint description: a *conjugate pair oracle* of the form in the following theorem.

**Theorem 1.** *Under the promise (4) of phase preservation of the quantum function oracle, we can write down the pair of maps in the computational basis and phase basis, respectively, as the conjugate pair oracle*

$$\mathcal{F}([x, \chi], [y, v]) = ([x, \chi + \check{f}(v)], [y + f(x), v]). \tag{16}$$

where  $\check{f}(v)$  is random with

$$\check{f}(v) = vT, \quad p(T = t) = |\delta_{t0} - 2\hat{f}(t)|^2 \tag{17}$$

so that  $\check{f}(v)$  is a Fourier sample of  $f$  when  $v = 1$ , thus the notation  $\check{f}(v)$  with a checkmark (the Kronecker delta function  $\delta_{t0}$  equals 1 when  $t$  equals the all-zero vector, and the Fourier transform used is the Hadamard transform).

**Proof.** The computational coordinate map immediately follows from the function map of the quantum oracle. The phase coordinate map is obtained from phase preservation (4) and the identity  $(-1)^w = 1 - 2w$  (skipping the steps already present in Equation (6)), through

$$\begin{aligned}
 H^{\otimes n+1}U_fH^{\otimes n+1}|\chi, v\rangle &= H^{\otimes n+1}\sum_{x,y}(-1)^{\chi\cdot x+v f(x)}|x\rangle(-1)^{vy}|y\rangle \\
 &= H^{\otimes n+1}\sum_{x,y}(-1)^{\chi\cdot x}(1-2vf(x))|x\rangle(-1)^{vy}|y\rangle \\
 &= \sum_{t,x}(-1)^{(\chi+t)\cdot x}(1-2vf(x))|x, v\rangle \\
 &= \sum_t(\delta_{t0}-2v\hat{f}(t))|\chi+t, v\rangle
 \end{aligned}
 \tag{18}$$

Therefore, the phase bit  $v$  of the target system is unchanged, while the phase bitstring  $\chi$  of the control is unchanged if  $v = 0$ , otherwise shifted with a random Fourier sample distributed as indicated in Equation (17).  $\square$

The above should be read as follows: if the input computational basis bitstrings  $x, y$  are well-defined, the output computational basis bitstrings  $x, y + f(x)$  are well-defined so that  $f(x)$  can be deduced. Furthermore, if instead, the input phase basis bitstrings  $\chi, v$  are well-defined, the output phase basis bitstrings  $\chi + \check{f}(v), v$  are well-defined so that  $\check{f}(v)$  can be deduced. By our definition of a conjugate pair, it is clear that only one part of the conjugate pair oracle  $\mathcal{F}$  can be queried at any one time.

In Fourier sampling, the distribution of  $\check{f}$  is especially simple. There is an additional promise on the structure of  $f$  in Equation (2), which in conjunction with phase preservation, using Theorem 1, gives

$$\check{f}(v) = vs \tag{19}$$

with probability 1. Here, we must stress that the simplicity of this is a direct consequence of phase preservation (Theorem 1) and the additional promise of Equation (2). We can now solve the Fourier Sampling problem by just accessing this function in the phase information, this gives us Algorithm 5.

---

**Algorithm 5** Conjugate Pair Oracle Fourier Sampling

---

**Conjugate pair oracle:**  $\mathcal{F}$

**Control:** none

**Bias/Target:** conjugate pair  $\mathcal{X}$

- 1: **create ancilla**  $\mathcal{Y} := [y, v]$  of length 1 with  $v := 1$
  - 2: **apply**  $\mathcal{F}$  to  $(\mathcal{X}, \mathcal{Y})$
  - 3: **discard ancilla**  $\mathcal{Y}$
- 

**Theorem 2.** For a target conjugate pair  $\mathcal{X} = [x, \chi]$  where the phase bitstring  $\chi = 0$ , Algorithm 5 solves Fourier Sampling. The solution is available after the algorithm as the phase bitstring  $\chi$ .

**Proof.** If the input bias conjugate pair  $\mathcal{X} = [x, \chi]$  has well-defined phase  $\chi = 0$ , it follows from Equation (19) that the output conjugate pair  $\mathcal{X} = [x, \chi]$  has well-defined phase  $\chi = s$  with probability 1.  $\square$

The output  $\chi$  is a bitstring of length  $n$ , so querying the phase part of the oracle can provide  $n$  bits of information in one use. This is obviously impossible for a classical oracle that returns a Boolean output.

Quantum Recursive Fourier Sampling can also be re-written from this point of view, adding some complication because of the  $k$  arguments  $x_j$ ; each phase coordinate  $\chi_j$  will now be shifted with a separate  $\check{f}_{k,j}$ . Theorem 1 generalizes in the following way.

**Theorem 3.** Under the promise (11) of phase preservation of the quantum function oracle, we can write down the pair of maps in the computational basis and phase basis, respectively, as the conjugate pair oracle

$$\begin{aligned} & \mathcal{F}_k([x_1, \chi_1], \dots, [x_k, \chi_k], [y, v]) \\ &= \left( [x_1, \chi_1 + \check{f}_{k,1}(v, x_2, \dots, x_k)], [x_2, \chi_2 + \check{f}_{k,2}(x_1, v, x_3, \dots, x_k)], \dots, \right. \\ & \quad \left. [x_k, \chi_k + \check{f}_{k,k}(x_1, \dots, x_{k-1}, v)], [y + f_k(x_1, \dots, x_k), v] \right). \end{aligned} \tag{20}$$

where  $\check{f}_{k,j}(x_1, \dots, x_{j-1}, v, x_{j+1}, \dots, x_k)$  is random with

$$\begin{aligned} & \check{f}_{k,j}(x_1, \dots, x_{j-1}, v, x_{j+1}, \dots, x_k) = vT_j(x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_k), \\ & p(T_j(x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_k) = t) = |\delta_{t0} - 2\widehat{f}_{k,j}(x_1, \dots, x_{j-1}, t, x_{j+1}, \dots, x_k)|^2. \end{aligned} \tag{21}$$

Here,  $\widehat{f}_{k,j}$  is the Fourier transform on the  $j$ :th argument of  $f_k$ , so that  $\check{f}_{k,j}$  is a Fourier sample on the  $j$ :th argument of  $f$  when  $v = 1$ .

**Proof.** For each  $j$ , use Theorem 1 on the conjugate pair oracle function  $\mathcal{F}_k(\mathcal{X}_1, \dots, \mathcal{X}_{j-1}, \mathcal{X}, \mathcal{X}_{j+1}, \dots, \mathcal{X}_k, \mathcal{Y})$ . Theorem 1 applies because this converts the phase preservation of Equation (11) into that of Equation (4).  $\square$

In RFS, we also have an additional promise on the structure of  $f_k$  in Equation (8), but only for the last argument, and in conjunction with phase preservation, using Theorem 3, we obtain

$$\check{f}_{k,k}(x_1, \dots, x_{k-1}, v) = v s_{k-1}(x_1, \dots, x_{k-1}) \tag{22}$$

with probability 1. Again, we must stress that the simplicity of this is a direct consequence of phase preservation (Theorem 3) and the additional promise of Equation (8).

The other  $\check{f}_{k,j}$  are random and depend on  $x_i, i \neq j$ , but we have no specific promised distribution when  $j < k$ ; this is simply not part of the problem description. Even so, we can solve Recursive Fourier Sampling by just accessing this function in the phase information of the  $k$ :th control. Note that a random value  $\check{f}_{k,j}, j < k$  is added to each  $\chi_j, j < k$  during this process, which needs to be removed to preserve the value of  $\chi_j$  when calculating the next level. This gives us Algorithm 6.

---

**Algorithm 6** Conjugate Pair Oracle Recursive Fourier Sampling

---

**Level:**  $1 \leq k \leq l$

**Conjugate pair oracles:**  $\mathcal{F}_{k+1}$  and  $\mathcal{G}_k$

**Controls:** conjugate pairs  $\mathcal{X}_j, 1 \leq j \leq k$

**Bias/Target:** conjugate pair  $\mathcal{Y}$

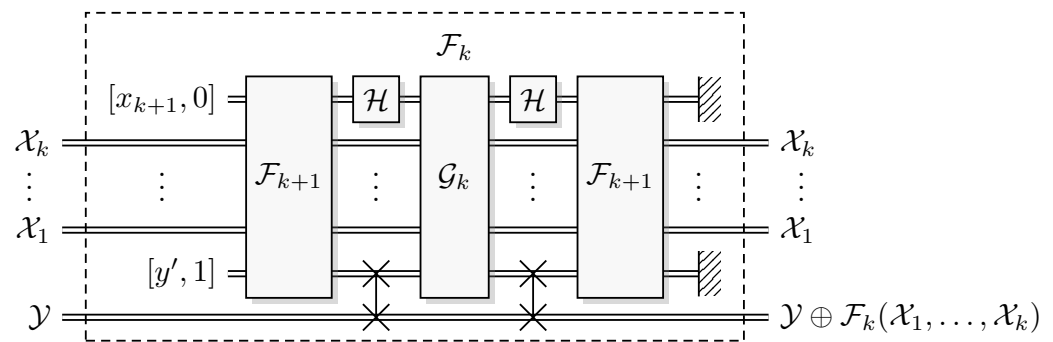
- 1: **create ancillas**  $\mathcal{X}_{k+1} = [x_{k+1}, \chi_{k+1}]$  with  $\chi_{k+1} := 0$  and  $\mathcal{Y}' = [y', v']$  with  $v' := 1$
  - 2: **apply**  $\mathcal{F}_{k+1}$  to  $(\mathcal{X}_1, \dots, \mathcal{X}_{k+1}, \mathcal{Y}')$
  - 3: **apply**  $\mathcal{H}^{n_{k+1}}$  to  $\mathcal{X}_{k+1}$
  - 4: **apply**  $\mathcal{G}_k$  to  $(\mathcal{X}_1, \dots, \mathcal{X}_{k+1}, \mathcal{Y})$
  - 5: **apply**  $\mathcal{H}^{n_{k+1}}$  to  $\mathcal{X}_{k+1}$
  - 6: **apply**  $\mathcal{F}_{k+1}$  to  $(\mathcal{X}_1, \dots, \mathcal{X}_{k+1}, \mathcal{Y}')$
  - 7: **discard ancillas**  $(\mathcal{X}_{k+1}, \mathcal{Y}')$
- 

**Theorem 4.** For a target conjugate pair  $\mathcal{Y} = [y, v]$  with phase bitstring  $v = 1$ , Algorithm 6 starting at level 1 solves Recursive Fourier Sampling. The solution is available after the algorithm as the phase bitstring  $\chi_1$ .

**Proof.** At level  $k$ , the proof has two parts: that oracle access to  $\mathcal{F}_{k+1}$  and  $\mathcal{G}_k$  give oracle access to  $\mathcal{F}_k$ ; and that the algorithm leaves the phase kick-back of  $\mathcal{F}_k$  in the arguments untouched. First, the ancillas  $(\mathcal{X}_{k+1}, \mathcal{Y}')$  have well-defined phases  $(0, 1)$ , so it follows from Equation (22) that  $\mathcal{X}_{k+1}$  has well-defined phase  $s_k(x_1, \dots, x_k)$  after applying  $\mathcal{F}_{k+1}$  the first time in step 2. The Hadamard of step 3 moves this into a well-defined computational basis value. It then follows from the promise (9) that oracle access to  $\mathcal{G}_k$  with this conjugate pair  $\mathcal{X}_{k+1}$  gives oracle access to  $\mathcal{F}_k$ , including the phase kickback map. Second, the transformation in step 6 is the exact inverse of the transformation in step 2, so the second addition will remove precisely the values added in step 2, so that  $\chi_j, j < k + 1$  only contains the phase kickback of  $\mathcal{F}_k$ .

That we have oracle access to  $\mathcal{F}_k$  for all  $k$  follows by induction, in particular, we obtain oracle access to  $\mathcal{F}_1$  so that we can deduce  $f_1(x_1)$  from  $\mathcal{F}_1([x_1, \chi_1], [y, v]) = ([x_1, \chi_1 + \check{f}_1(v)], [y + f_1(x_1), v])$  by using the well-defined input  $x_1$  and bias  $y = 0$ .  $\square$

The crucial point here is that in this formulation of the problem and solution algorithm, although technically equivalent to the quantum formulation, steps 5 and 6 are no longer motivated by the somewhat unclear need to “uncompute ‘garbage’ [in the computational coordinate of the ancillary systems] left over by the first call, and thereby enable [proper] interference” [2,16]. Here, the motivation is much clearer: We need to uncompute the shift of the phase coordinate of the controlling systems with indices  $j < k + 1$ , i.e., uncompute phase coordinate garbage, see Figure 2. This enables a direct comparison with classical reversible computing where (computational coordinate) garbage is uncomputed to enable reversibility. In RFS, the uncomputation of phase coordinate garbage enables the quantum advantage.



**Figure 2.** Conjugate Pair Recursive Fourier Sampling. The focus of the uncomputation is to undo the shift of the phase coordinate of  $\mathcal{X}_1, \dots, \mathcal{X}_k$  from the  $\mathcal{F}_{k+1}$  oracle.

**Corollary 1.** When using Algorithm 6 to solve Recursive Fourier Sampling, the uncomputation in step 6 is necessary to uncompute phase coordinate garbage added to  $\mathcal{X}_j, j < k + 1$  in step 2.

**Proof.** Step 2 of Algorithm 6 adds random values to the phase bitstrings of  $\mathcal{X}_j, j < k + 1$  according to Equation (20). Step 6 is necessary to subtract values that are identical to the added values.  $\square$

The reason we need to perform uncomputation, to invert the addition of these additional phases, is that we are not guaranteed anything about the nature of  $\check{f}_{k,j}, j < k + 1$ , by the problem formulation. Adding such a guarantee in the problem statement could give a further quantum advantage, although that would require maintaining the classical complexity of the problem under such an addition. One possible addition is promising linear  $g_k$  at every level, but as mentioned earlier, that would reduce the classical complexity considerably [1].

## 5. Conclusions

In this work, the main contribution is the notion of conjugate pair oracles, which enables a reformulation of RFS, in turn, providing a stronger argument for why uncomputation is necessary than previously available. We also provide a small generalization of Recursive Fourier Sampling: our generalized RFS contains both the original formulation by Bernstein and Vazirani [3] and the one by Aaronson [16] as special cases.

Uncomputation is needed because the function oracle (quantum or conjugate pair, both with phase kickback) adds random phase shifts, i.e., computational garbage, to the controlling registers. Furthermore, we are not guaranteed anything about the value or distribution of these phase shifts by the problem formulation; the only guarantee we have concerns the phase shift of the last argument. Adding further guarantees in the problem statement of the behavior of the involved functions could give a further quantum advantage, although that would require maintaining the classical complexity of the problem under such an addition; we conjecture that such a modification is possible.

It should be noted here that the complexity bound on solving RFS, i.e., the exact bound  $\Theta(2^l)$  mentioned at the end of Section 3 [4,16], already indicates that uncomputation is needed. However, phase kickback in the conjugate pair oracle paradigm presented here arguably gives a better understanding of the necessity of uncomputation of phase coordinate garbage within the Quantum RFS algorithm.

It has previously been established [7] that at least some problems in NP relative to an oracle (such as Simon's algorithm [2]) use phase kickback as their driving mechanism. We now have that RFS, which in a very real sense is "more difficult" than Simon's algorithm given that RFS lies outside MA relative to an oracle [3], relies on this very same property of quantum systems as the enabling computational resource. That is, in the oracle paradigm, it is not so much a matter of computational power enabling an advantage, but rather one of communication: accessibility of the relevant information outside the oracle. As long as the correct phase information is accessible, we can (apparently) solve extremely hard problems more efficiently than otherwise. This strengthens the argument [7] that phase kickback is an important property of quantum computational systems. Clearly, phase kickback is critical in enabling a quantum advantage in RFS.

This formalism gives a new understanding of the behavior of quantum algorithms such as Grover's algorithm [18] or Shor's algorithm [19], in that it allows the explicit tracking of the phase information in a quantum circuit [7,10]. It also points out a possible path towards proving a practically meaningful distinction between classical and quantum computation. Perhaps the difficulty in classically simulating quantum computation stems entirely from the difficulty in maintaining the correct phase map in a quantum computation. Investigating this venue more thoroughly may well lead to the unconditional separation theorem that the quantum information society is still striving for to this day.

**Author Contributions:** Conceptualization, C.H. and J.-Å.L.; methodology, C.H., N.J. and J.-Å.L.; validation, C.H. and J.-Å.L.; formal analysis, C.H., N.J. and J.-Å.L.; investigation, C.H. and J.-Å.L.; writing—original draft preparation, C.H.; writing—review and editing, C.H., N.J. and J.-Å.L.; visualization, C.H. and J.-Å.L.; supervision, J.-Å.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was partially funded by the Swedish Research Council through grant agreement no. 2023-05031.

**Data Availability Statement:** No new data were created or analyzed in this study. Data sharing is not applicable to this article.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Appendix A. Previous Versions of RFS

In our formulation, there is no connection between the length  $n_k$  of individual bit-strings  $x_k$  and the total recursion depth  $l$ . This is not the case in [3], where given  $n_1 = n$ , the recursion depth  $l = \log n$ , since there  $n_k = n/2^{k-1}$  decrease exponentially. Our approach also generalizes the formulation of Aaronson [16] where  $n_k = n$  and the nonlinear output function  $g$  are the same for all recursion levels. Both of these are special cases of our formulation, and so any algorithm and accompanying circuit that solves the latter also solves the former.

For completeness, we now give an explicit description of how to move between the notation of Aaronson [16] and the one used here. Note that the index order for  $x_j$  is reversed with respect to Ref. [16]; we here also write uppercase  $G$  for the nonlinear output function of Ref. [16] to distinguish it from  $g_k$  as used here. Aaronson [16] defines height- $h$  Recursive Fourier Sampling, or  $\text{RFS}_h$ , recursively as follows. We are given oracle access to a function  $A(x_1, \dots, x_h)$  for all  $x_1, \dots, x_h \in \{0, 1\}^n$ , and are promised the following:

- (i) For each fixed  $x_h^*$ ,  $A(x_1, \dots, x_{h-1}, x_h^*)$  is an instance of  $\text{RFS}_{h-1}$  on  $x_1, \dots, x_{h-1}$ , having answer bit  $b(x_h^*) \in \{0, 1\}$ ;
- (ii) There exists a secret string  $s \in \{0, 1\}^n$  such that  $b(x_h^*) = s \cdot x_h^*$  for each  $x_h^*$ .

The answer bit to be returned is  $G(s)$ .

Given such an instance of  $\text{RFS}_{l+1}$ , we can rewrite it as a Fourier Sampling tree  $\{f_k, 1 \leq k \leq l + 1\}$  derived from a sequence  $g_k$  as follows. First, for each  $k$  we let the function  $f_k$  equal the output  $A$  of the  $\text{RFS}_k$  instance obtained recursively from promise (i). For  $k > 1$ , we find from promise (ii) that

$$f_k(x_1, \dots, x_{k-1}, x_k^*) = A(x_1, \dots, x_{k-1}, x_k^*) = b(x_k^*) = s \cdot x_k^*, \tag{A1}$$

making  $f_k$  a Fourier Sampling tree. We should point out here that  $s$  will actually depend on the free parameters  $x_1, \dots, x_{k-1}$ , which correspond to the explicit arguments of  $s_{k-1}(x_1, \dots, x_{k-1})$  in Equations (8) and (9). We also know that the function output from  $\text{RFS}_{k-1}$  is

$$f_{k-1}(x_1, \dots, x_{k-1}) = A(x_1, \dots, x_{k-1}) = G(s_{k-1}(x_1, \dots, x_{k-1})), \tag{A2}$$

so, if we let

$$g_{k-1}(x_1, \dots, x_k) = G(x_k), \tag{A3}$$

the Fourier Sampling tree  $f_k$  is derived from the sequence  $g_k$ . Oracle access to  $A(x_1, \dots, x_{l+1})$  and  $G(x_k)$  gives oracle access to  $f_{l+1}(x_1, \dots, x_{l+1})$  and  $g_{k-1}(x_1, \dots, x_k)$  for all  $k$ , respectively.

Conversely, it is possible to rewrite a Fourier Sampling tree  $f_k$  derived from  $g_k$  as a recursively defined sequence of  $\text{RFS}_k$  instances, if all  $n_k = n$  and all  $g_{k-1}(x_1, \dots, x_k)$  equal one and the same function of the last argument. Then, to create an  $\text{RFS}_h$  instance, we let  $A(x_1, \dots, x_h) = f_h(x_1, \dots, x_h)$ , and  $G(x) = g_1(0, x) = \dots = g_l(0, \dots, 0, x)$ , which immediately fulfill promises (i) and (ii). Oracle access to  $f_{l+1}(x_1, \dots, x_{l+1})$  and  $g_1(0, x)$  gives oracle access to  $A(x_1, \dots, x_{l+1})$  and  $G(x)$ , respectively.

## References

1. Bernstein, E.; Vazirani, U. Quantum complexity theory. In Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC '93), San Diego, CA, USA, 16–18 May 1993; ACM: New York, NY, USA, 1993; pp. 11–20. [CrossRef]
2. Bernstein, E.; Vazirani, U. Quantum Complexity Theory. *SIAM J. Comput.* **1997**, *26*, 1411–1473. [CrossRef]
3. Vazirani, U. A Survey of Quantum Complexity Theory. In *Proceedings of Symposia in Applied Mathematics*; AMS: Providence, RI, USA, 2002; Volume 58, pp. 193–217. [CrossRef]

4. Johnson, B.E. Upper and Lower Bounds for Recursive Fourier Sampling. Ph.D. Thesis, University California Berkeley, Berkeley, CA, USA, 2008.
5. Cleve, R.; Ekert, A.; Macchiavello, C.; Mosca, M. Quantum algorithms revisited. *Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci.* **1998**, *454*, 339–354. [[CrossRef](#)]
6. Nielsen, M.A.; Chuang, I.L. *Quantum Computation and Quantum Information*; Volume 10th Anniversary Edition; Cambridge University Press: Cambridge, UK, 2010.
7. Johansson, N.; Larsson, J.Å. Quantum Simulation Logic, Oracles, and the Quantum Advantage. *Entropy* **2019**, *21*, 800. [[CrossRef](#)] [[PubMed](#)]
8. Wootters, W.K. Picturing Qubits in Phase Space. *arXiv* **2003**, arXiv:quant-ph/0306135. [[CrossRef](#)]
9. Aaronson, S.; Gottesman, D. Improved Simulation of Stabilizer Circuits. *Phys. Rev. A* **2004**, *70*, 052328. [[CrossRef](#)]
10. Hindlycke, C.; Larsson, J.Å. Efficient Contextual Ontological Model of  $n$ -Qubit Stabilizer Quantum Mechanics. *Phys. Rev. Lett.* **2022**, *129*, 130401. [[CrossRef](#)] [[PubMed](#)]
11. Spekkens, R.W. Evidence for the Epistemic View of Quantum States: A Toy Theory. *Phys. Rev. A* **2007**, *75*, 032110. [[CrossRef](#)]
12. Gottesman, D. Theory of Fault-Tolerant Quantum Computation. *Phys. Rev. A* **1998**, *57*, 127–137. [[CrossRef](#)]
13. Shor, P. Fault-Tolerant Quantum Computation. In Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Burlington, VT, USA, 14–16 October 1996; pp. 56–65. [[CrossRef](#)]
14. Calderbank, A.R.; Rains, E.M.; Shor, P.W.; Sloane, N.J.A. Quantum Error Correction and Orthogonal Geometry. *Phys. Rev. Lett.* **1997**, *78*, 405–408. [[CrossRef](#)]
15. Bravyi, S.; Gosset, D.; König, R. Quantum Advantage with Shallow Circuits. *Science* **2018**, *362*, 308–311. [[CrossRef](#)] [[PubMed](#)]
16. Aaronson, S. Quantum Lower Bound for Recursive Fourier Sampling. *Quantum Inf. Comput.* **2003**, *3*, 165–174. [[CrossRef](#)]
17. McKague, M. Interactive proofs with efficient quantum prover for recursive Fourier sampling. *Chic. J. Theor. Comput. Sci.* **2012**, *18*, 1–10. [[CrossRef](#)]
18. Grover, L.K. A Fast Quantum Mechanical Algorithm for Database Search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; ACM: New York, NY, USA, 1996; STOC '96; pp. 212–219. [[CrossRef](#)]
19. Shor, P. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 20–22 November 1994; pp. 124–134. [[CrossRef](#)]

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.