Quantum matching pursuit: A quantum algorithm for sparse representations

Armando Bellante* and Stefano Zanero
Politecnico di Milano, DEIB, Via Ponzio 34/5 Building 20, 20133 Milan, Italy

(Received 10 August 2021; accepted 25 January 2022; published 11 February 2022)

Representing signals with sparse vectors has a wide spectrum of applications that ranges from image and video coding to shape representation and health monitoring. In many applications with real-time requirements or that deal with high-dimensional signals, the computational complexity of the encoder that finds the sparse representation plays an important role. Quantum computing has recently shown promising speedups in many representation learning tasks. In this work, we propose a quantum version of the well-known matching-pursuit algorithm. Assuming the availability of a fault-tolerant quantum random access memory, our quantum matching pursuit lowers the complexity of its classical counterpart by a polynomial factor, at the cost of some error in the computation of the inner products, enabling the computation of sparse representations of high-dimensional signals. Besides proving the computational complexity of our algorithm, we provide numerical experiments that show that its error is negligible in practice. This work opens the path to further research on quantum algorithms for finding sparse representations, showing suitable quantum computing applications in signal processing.

DOI: 10.1103/PhysRevA.105.022414

I. INTRODUCTION

Finding a sparse representation is the problem of representing a dense signal as a linear combination of a few unit vectors, also referred to as *atoms*. Usually, the set of atoms is larger than the space where the signal lies, as overcomplete sets of atoms enable sparser representations [1]. Once a set of atoms, or *dictionary*, is fixed, the sparse representation of the signal is the set of coefficients of their linear combination.

Signals of the same type are likely to be represented sparsely over the same dictionary. For instance, the widely used JPEG algorithm exploits the fact that images are sparse with respect to the discrete-cosine-transform basis to perform compression [2]. Finding sparse representations is a subject of interest in many fields, and its applications range from data compression to denoising and anomaly detection [3,4].

When these applications have real-time requirements or deal with high-dimensional signals, the computational cost of finding the representation is crucial. Unfortunately, finding the sparsest representation that approximates the signal is an NP-hard problem and is intractable in practice. For this reason, researchers have developed a series of greedy algorithms that, through local optimizations, find approximate solutions in an acceptable running time.

In recent years, the effectiveness of quantum computing in representation learning has become increasingly evident. Recent research has proven computational advantages for algorithms such as principal component analysis [5], slow feature analysis [6], and spectral clustering [7].

In this work, we propose an end-to-end quantum algorithm for learning sparse representations using a matching-pursuit approach. We develop a quantum version of the well-known matching-pursuit algorithm [8], reducing its computational cost by a polynomial factor. While there are some known speedups in the case of specific analytical dictionaries [9], to our knowledge, there are no classical algorithms that compare with our run time over a general dictionary.

Besides thoroughly analyzing the running time and error of our algorithm, we describe a suitable quantum processing framework that can be used as a starting point to construct other quantum pursuit algorithms.

The remainder of this paper is organized as follows. In Sec. II, we discuss previous work that relates to ours. Section III describes the classical matching-pursuit algorithm, introducing the necessary notation for both the quantum and classical versions. In Sec. IV, we briefly introduce the concept of quantum computation and some subroutines that will serve as a basis for the quantum algorithm. Section V presents the quantum matching-pursuit algorithm, providing a thorough run-time and error analysis. Finally, in Sec. VI, we run numerical experiments to show that the quantum matching pursuit can find representations that are as sparse as those of its classical counterpart.

II. RELATED WORK

The matching-pursuit algorithm was first introduced by Mallat and Zhang [8]. The original version of the algorithm has a running time of $O(knm)$, where k is the number of optimization iterations, n is the length of the signal, and m is the number of atoms in the dictionary.

Among the attempts to speed up the matching-pursuit algorithm, the closest to ours is the one of Krstulovic and Gribonval [9]. They exploited particular properties of some analytic (nonlearned) dictionaries, like the multiscale time-frequency Gabor dictionary [8], to reduce the run time of

*armando.bellante@polimi.it

matching pursuit to $O(kn\log(n))$. However, their algorithm is still slow on nonanalytical dictionaries.

While some previous works suggest the use of matching pursuit to simulate the dynamics of quantum-mechanical processes [10–13], to our knowledge there is no previous work that discusses quantum speedups for finding sparse representations of signals over large dictionaries.

III. CLASSICAL MATCHING PURSUIT

A. Notation

We denote matrices using capital letters and use lowercase letters for vectors and scalars. Given a matrix A, its ith row and column are denoted by $a_{i,\cdot}$ and a_i, respectively. The component identified by the ith row and the jth column is denoted a_{ij}. We write the jth element of a vector u as u_j.

Let $x \in \mathbb{C}^n$ be a unit vector. Using Dirac's notation, we use $|x\rangle$ to represent it as a column vector and $\langle x|$ to denote its complex-conjugate row vector. We use $\langle a_i, b_j\rangle$ to denote the inner product between two vectors a_i and b_j.

The notation $\|\cdot\|_2$ indicates the Euclidean norm of a vector. The pseudonorm $\|\cdot\|_0$ is the number of nonzero components of a vector. The symbol $\|\cdot\|_F$ indicates the Frobenius norm of a matrix. For a matrix $A \in \mathbb{R}^{n\times m}$, the Frobenius norm is defined as $\|A\|_F = \sqrt{\sum_i^n \sum_j^m a_{ij}^2}$.

When stating the complexity of an algorithm, the $\widetilde{O}(\cdot)$ notation omits polylogarithmic terms in the input data size [e.g., if an algorithm uses a matrix $A \in \mathbb{R}^{n\times m}$, $\widetilde{O}(1) \equiv O(\text{polylog}(nm))$].

B. Problem statement

We can represent a signal as a vector $s \in \mathbb{R}^n$. We use d_j to denote the jth atom over which we search the sparse representation. Each atom is a unit vector, meaning that for every j we have $\|d_j\|_2 = 1$. A dictionary is a matrix $D \in \mathbb{R}^{n\times m}$ whose columns are the atoms d_j for $j \in \{0, \ldots, m-1\}$. In most of the interesting cases, the dictionary is overcomplete (i.e., $m > n$).

Formally, given a signal $s \in \mathbb{R}^n$ and a dictionary $D \in \mathbb{R}^{n\times m}$, the problem of finding a sparse representation $x \in \mathbb{R}^m$ of the signal is known as $\mathcal{P}_0^\epsilon$.

Definition 1. Problem $\mathcal{P}_0^\epsilon$. Given $s \in \mathbb{R}^n$, $D \in \mathbb{R}^{n\times m}$, and $\epsilon \in \mathbb{R}^+$, problem $\mathcal{P}_0^\epsilon$ is defined as

$$\arg\min_x \|x\|_0 \text{ such that } \|Dx - s\|_2 \leqslant \epsilon. \tag{1}$$

Finding the exact solution to problem $\mathcal{P}_0^\epsilon$ is an NP-hard task [14]. While quantum computers are not expected to solve NP-hard problems in polynomial time (indeed, it is widely believed that NP $\not\subseteq$ BQP [15]), they can still provide speedups of practical use on greedy algorithms that compute approximate solutions. In this paper we propose a quantum version of the matching-pursuit algorithm, a greedy approach to approximately solve the $\mathcal{P}_0^\epsilon$ problem in polynomial time.

C. Algorithm

The strategy behind the matching-pursuit algorithm is to face the problem through subsequent optimization steps. Starting from an empty solution $x = 0^{\otimes m}$, the matching pursuit searches for the atom that best reduces the difference between the representation Dx and the signal at each iteration, updating the solution iteratively. We now discuss the matching-pursuit algorithm in detail.

As an initialization step, we create a residual vector and an empty solution

$$r = s, \tag{2}$$

$$x = 0^{\otimes m}. \tag{3}$$

Since we are at the beginning of the algorithm and $s - Dx = s$, the residual is set equal to the signal.

Once the initialization is complete, the algorithm searches for the atom closest to the residual by computing

$$j^* = \arg\min_j \|r - z_j d_j\|_2, \tag{4}$$

where $z_j \in \mathbb{R}$ is the best scaling factor for the atom d_j,

$$z_j = \arg\min_z \|r - z d_j\|. \tag{5}$$

It is possible to show that $z_j = \langle r, d_j\rangle$ [8], from which we can derive the following equivalence:

$$\|r - z_j d_j\|_2^2 = \|r\|_2^2 - |\langle r, d_j\rangle|^2. \tag{6}$$

Because of Eq. (6), finding the best atom [Eq. (4)] is equivalent to searching for the maximum absolute value of the inner products between the current residual and the atoms

$$j^* = \arg\max_j |\langle r, d_j\rangle|. \tag{7}$$

This step is known as the *sweep stage*, as we need to iterate over all the atoms in the dictionary to compute the inner products and choose the best one.

After selecting the best atom, both the solution and the residual get updated,

$$x_{j^*} = x_{j^*} + z_{j^*}, \tag{8}$$

$$r = r + z_{j^*} d_{j^*}. \tag{9}$$

Updating the residual makes it so that the algorithm does not consider the part of the signal that has been modeled so far. At each iteration, the residual is $r = s - Dx$.

The algorithm continuously searches for the best approximating atom and performs the updates until the following stopping condition is met:

$$\|x\|_0 > L \quad \text{or} \quad \|r\|_2 \leqslant \epsilon \tag{10}$$

for a sparsity threshold $L \in \mathbb{N}^+$ and an error reconstruction tolerance $\epsilon \in \mathbb{R}^+$. We remark that, at each iteration, the norm of the residual $\|r\|_2 = \|s - Dx\|_2$ expresses how well our solution approximates the original signal.

We summarize this procedure in Algorithm 1.

D. Computational complexity

The analysis of the run time of the algorithm proceeds as follows. The initialization step is linear in the length of the residual $O(n)$. The computation of the sweep stage (steps 3–5) is the bottleneck of this algorithm. Indeed, the algorithm computes m inner products of vectors of length n, which requires time $O(nm)$. Selecting the best atom has a negligible cost, as

Algorithm 1. Matching pursuit.

Initialize $r = s$, $x = 0^{\otimes m}$.
while not ($\|x\|_0 > L$ or $\|r\|_2 \leqslant \epsilon$) **do**
  for all $j \in [m]$ **do**
    Compute $\langle d_j, r\rangle$
  end for
  Select $j^* = \text{argmax}(|\langle d_j, r\rangle|)$
  Assign $z = \langle d_{j^*}, r\rangle$
  Update the solution $x_j = x_j + z$
  Update the residual $r = r - z d_{j^*}$
end while
Output x.

it can be done during the computation of the inner products without significant overhead. Updating the solution is $O(1)$, and the residual's update is bounded by $O(n)$.

The complexity of the sweep stage dominates all the other complexities in the loop. Therefore, assuming that the matching pursuit converges after k iterations, its asymptotic computational complexity scales as

$$O(knm). \tag{11}$$

IV. QUANTUM COMPUTING BACKGROUND

A. Quantum computation

Just like a bit is the fundamental unit of information in classical computing, a qubit is the fundamental information unit in quantum computing. A qubit is a mathematical representation of a quantum-mechanical object and can be described as an ℓ_2 normalized vector of $\mathbb{C}^2$. The state of an n-qubit system (a register of a quantum computer) is the tensor product of single qubits: a unitary vector $|x\rangle \in H^{\otimes n} \simeq \mathbb{C}^{2^n}$. In other words, with $|i\rangle \in H^{\otimes n}$ we denote a quantum register that contains the binary expansion of number i. Its corresponding complex vector is a vector of length 2^n, full of zeros, with the ith element equal to one. For instance,

$$|3\rangle \in H^{\otimes 2} = |1\rangle\,|1\rangle = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}. \tag{12}$$

Given a basis $\{|i\rangle\}_0^{n-1}$ for $H^{\otimes \log_2(n)}$, with $\log_2(n)$ qubits, we can describe a quantum state $|\psi\rangle = \sum_i^n \alpha_i |i\rangle$ with $\sum_i^n |\alpha_i|^2 = 1$. The values $\alpha_i \in \mathbb{C}$ are called amplitudes of the quantum states $|i\rangle$ for state $|\psi\rangle$.

The evolution of a quantum system is described by unitary matrices U. Unitary matrices are norm preserving and thus can be used as a suitable mathematical description of pure quantum evolutions. Any quantum algorithm that does not perform measurements can be represented by a unitary matrix.

Quantum states can be measured, but measurements alter the state. In this work, measurements are performed with respect to the computational basis $\{|i\rangle\}_0^{n-1}$ of $H^{\otimes \log_2(n)}$. This means that if we measure a quantum register $|\psi\rangle = \sum_i^n \alpha_i |i\rangle$, it can collapse to any state $|i\rangle$, each with probability $|\alpha_i|^2$. It is important to recall that no quantum algorithm can create a copy of a generic quantum state. Therefore, to measure a state multiple times, it is necessary to create it again from scratch every time.

For a deeper introduction to the subject, we encourage the reader to consult Nielsen and Chuang [16].

B. Quantum random access memory

A QRAM, or quantum random access memory (RAM), is a device that, given a list of n bit strings $x_i \in \{0,1\}^m$ of length m, performs the following mapping in time $O(\text{polylog}(n))$:

$$\sum_i^n \alpha_i |i\rangle\,|0\rangle \mapsto \sum_i^n \alpha_i |i\rangle\,|x_i\rangle\,, \tag{13}$$

where $\alpha_i \geqslant 0$ and $|x_i\rangle \in H^{\otimes m}$ is the state of the computational basis that corresponds to the bit string x_i.

The reader can think of it as a quantum equivalent of the classical RAM. In a classical RAM, we can store n values and query any of those in time $O(1)$, considering it can access the m bits in parallel. The main difference from a classical RAM is that a quantum RAM needs to perform queries in superposition.

As explained in the next section, our quantum algorithm, like many previous ones, assumes the availability of such a device to encode scalars, matrices, and vectors in quantum states efficiently.

Building a fault-tolerant, hardware-efficient QRAM is not an easy task. One of the most promising proposals to structure the quantum random access memory is the bucket-brigade architecture. First presented in Giovannetti *et al.* [17], this architecture is composed of $O(n)$ gates, while its circuit is only $O(\log(n))$ deep. Current error-analysis research claims that algorithms that query the bucket-brigade QRAM a super-polylogarithmic number of times (e.g., Grover's search and, consequently, ours) likely require the bucket-brigade QRAM to be error corrected [18,19]. This requires additional hardware and created skepticism of the effective speedup of this class of algorithms in this input model [18]. Recently, Hann *et al.* [19] showed that the bucket-brigade architecture is highly resilient to generic errors and that its architectural advantage persists even in the case of error correction, contrary to what was previously believed. At the same time, a recent work showed how to build a fault-tolerant bucket-brigade QRAM by parallelizing Clifford + T gates [20], at the cost of using $O(n)$ ancillary qubits.

In this work, we will perform our analysis assuming access to a fault-tolerant QRAM, capable of performing queries in time $O(\text{polylog}(n)) \sim \widetilde{O}(1)$, with n being the number of entries stored in the QRAM. With this assumption in mind, we can explain our data-access model.

C. Quantum data access

We can encode a scalar $a \in \mathbb{R}$ in a quantum register $|a\rangle$ using its binary encoding and retrieve it by measuring the register in the computational basis, just as discussed in the previous section. To encode a scalar, we need as many qubits as the classical bits that store it. For more detailed information on how to encode a real number in a quantum register, we suggest reading Nielsen and Chuang [16] (see Chap. 5, Secs. 5.1 and 5.2) to see how phase angles are encoded in a

state and Rebentrost *et al.* [21] for a more formal definition of a quantum arithmetic model with fixed point precision.

On the other hand, the components of a vector $a \in \mathbb{R}^m$ can be encoded with fewer qubits than classical bits, using the amplitudes of a quantum state. When measured, the quantum state collapses to an index of the vector with probability proportional to the magnitude of the indexed component. We call this representation state vector.

Definition 2. State vector. Given a vector $x \in \mathbb{R}^n$, the corresponding state vector is the following quantum state: $|x\rangle = \frac{1}{\|x\|_2}\sum_i^n x_i |i\rangle$, which is encoded in $\lceil \log n \rceil$ qubits.

We say we have quantum access to a classical vector $x \in \mathbb{R}^n$ if we have access to a unitary operator that performs the mapping $U_x : |0\rangle \mapsto |x\rangle$ in time $O(log(n))$. Similarly, we define the concept of quantum access to a matrix.

Definition 3. Quantum access to a matrix. We say we have quantum access to a matrix $A \in \mathbb{R}^{n\times m}$ if we can perform the following mappings in time $O(\text{polylog}(nm))$:

$$U : |i\rangle |0\rangle \mapsto |i\rangle |a_{i,\cdot}\rangle = |i\rangle \frac{1}{||a_{i,\cdot}||}\sum_j^m a_{ij} |j\rangle$$

for $i \in \mathbb{R}^n$;

$$V : |0\rangle \mapsto \frac{1}{\|A\|_F}\sum_i^n \|a_{i,\cdot}\| \, |i\rangle .$$

By combining the two unitaries above, it is possible to represent a matrix in a quantum state

$$|A\rangle = U(V \otimes \mathbb{I}) |0\rangle |0\rangle = \frac{1}{\|A\|_F}\sum_i^n\sum_j^m a_{ij} |i\rangle |j\rangle \quad (14)$$

$$= \frac{1}{\|A\|_F}\sum_i^n \|a_{i,\cdot}\| \, |i\rangle |a_{i,\cdot}\rangle \quad (15)$$

using two registers of $\lceil \log(n) \rceil + \lceil \log(m) \rceil$ qubits.

The Appendix of Kerenidis and Prakash [22] shows in detail how to construct a classical data structure that enables efficient computation of the unitary matrices that give quantum access to vectors and matrices. This classical data structure can be created in $O(nm \log^2(nm))$ for an $n \times m$ matrix and in $O(n \log(n))$ for a vector of length n. Assuming the ability to perform quantum queries on the entries of this data structure in superposition (i.e., assuming the availability of a QRAM that stores the entries of these trees), the authors show how to provide quantum access to a vector or matrix in time $\widetilde{O}(1)$.

In practice, in this input model, at the cost of some classical preprocessing, it is possible to encode vectors and matrices in quantum states using a small number of qubits in time $\widetilde{O}(1)$.

D. Relevant quantum subroutines

We now introduce two quantum subroutines that are particularly important to our work. By slightly modifying these two results, we introduce two new corollaries that are more suitable to our needs.

Lemma 1. Inner product estimation [23]. Let there be quantum access to the matrices $V \in \mathbb{R}^{n\times m}$ and $C \in \mathbb{R}^{k\times m}$, through the unitaries $U_v : |i\rangle |0\rangle \mapsto |i\rangle |v_{i,\cdot}\rangle$ and $U_c : |j\rangle |0\rangle \mapsto |j\rangle |c_{j,\cdot}\rangle$, that run in time T. Then, for any $\delta > 0$ and $\epsilon > 0$, there exists a quantum algorithm that computes $|i\rangle |j\rangle |0\rangle \mapsto |i\rangle |j\rangle |\overline{\langle v_{i,\cdot}, c_{j,\cdot}\rangle}\rangle$, such that $|\overline{\langle v_{i,\cdot}, c_{j,\cdot}\rangle} - \langle v_{i,\cdot}, c_{j,\cdot}\rangle| \leqslant \epsilon$, with probability greater than $1 - 2\delta$ in time $\widetilde{O}(\frac{T \log(1/\delta)}{\epsilon})$.

In our quantum matching pursuit we will need to perform inner products between the columns of a matrix and a vector. It is possible to use this result to prepare a state that stores the inner products between the columns of a matrix A and a column vector x. Indeed, we need quantum access only to the matrix's transpose A^T and to the vector x via unitaries U_A and U_x. If we have that, the two unitaries that prepare the states trivially become $U_v = U_{A^T}$ and $U_c = (\mathbb{I} \otimes U_x)$, and we can ignore the second register. We also stress that having quantum access to a matrix is equivalent to having access to its transpose. Indeed, if we swap the first and the second registers of Eq. (14), we have a quantum state that represents the transposed matrix. Finally, using consistent phase estimation from Ta-Shma [24], it is possible to modify the algorithm above so that the error across several runs is consistent.

Corollary 1. Matrix-vector product estimation. Let there be quantum access to a matrix $A \in \mathbb{R}^{n\times m}$ and to a unit vector $x \in \mathbb{R}^m$, through the unitaries $U_A : |0\rangle |0\rangle \mapsto \frac{1}{\|A\|_F}\sum_i^n |i\rangle |a_{i,\cdot}\rangle$ and $U_x : |0\rangle \mapsto \frac{1}{\|x\|_2}\sum_i^m x_i |i\rangle$, that run in time less than T. Then, for any $\delta > 0$ and $\epsilon > 0$, there exists a quantum algorithm that computes $|0\rangle |0\rangle \mapsto \frac{1}{\|A\|_F}\sum_i^n |i\rangle |\overline{\langle a_{i,\cdot}, x\rangle}\rangle$, such that $|\overline{\langle a_{i,\cdot}, x\rangle} - \langle a_{i,\cdot}, x\rangle| \leqslant \epsilon$ consistently across multiple runs, with probability greater than $1 - 2\delta$ in time $\widetilde{O}(\frac{T \log(m/\delta)}{\epsilon})$.

To prove the bound on the success probability of the corollary above, we can exploit the union bound, also known as Boole's inequality,

$$P\big(\cup_i^m p_f(i)\big) \leqslant \sum_i^m p_f(i). \quad (16)$$

The union bound states that, given a set of likely events, the probability that any one of them happens is lower than the sum of their individual probabilities.

The run-time overhead of Lemma 1 to bound the failure probability of one inner product with $p_f(i) \leqslant 2\delta$ is $O(\log(1/\delta))$. Bounding the failure probability of one product by $p_f(i) \leqslant 2\delta'$, the probability P_f that one of the m inner products will fail can be bounded by

$$P_f \leqslant \sum_i^m p_f(i) \leqslant 2m\delta'. \quad (17)$$

Choosing $\delta' = \frac{\delta}{m}$, we have that the algorithm succeeds with probability $1 - 2\delta$ with a run-time overhead of $O(\log(\frac{m}{\delta}))$.

Finally, if x is not a unit vector, we can multiply the result by $\|x\|_2$ to get an estimate within $\epsilon\|x\|_2$ error.

The second algorithm that we introduce enables searching for the minimum value of an unsorted array quadratically faster than what we can do classically.

Lemma 2. Finding the minimum [25]. Let there be quantum access to a vector $u \in [0, 1]^N$ via the operation $|j\rangle |0\rangle \to |j\rangle |u_j\rangle$ in time T. Then, we can find the minimum $u_{\min} = \min_{j\in[N]} u_j$ and its index $j_{\min} = \arg\min_{j\in[N]} u_j$ with success probability $1 - \delta$ in time $O(T\sqrt{N}\log(\frac{1}{\delta}))$.

This algorithm is built around the famous Grover's search algorithm [26]. Grover's algorithm takes advantage of an oracle to mark the elements of the superposition that satisfy the search conditions. An oracle is a function $f : \mathbb{R} \mapsto \{0, 1\}$. The "finding the minimum" routine uses Grover's search many times, using oracles of the type

$$f_i(j) = \begin{cases} 1 & \text{if } u_j < u_i, \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

By using the same algorithm described in Durr and Hoyer [25] with oracles

$$f_i'(j) = \begin{cases} 1 & \text{if } |u_j| > |u_i|, \\ 0 & \text{otherwise,} \end{cases} \tag{19}$$

it is possible to find the maximum absolute value of an array in the same running time.

Corollary 2. Finding the maximum absolute value. Let there be quantum access to a vector $u \in [0, 1]^N$ via the operation $|j\rangle |0\rangle \to |j\rangle |u_j\rangle$ in time T. Then, we can find the maximum absolute value $u_{\max} = \max_{j\in[N]} |u_j|$ and its index $j_{\min} = \arg\min_{j\in[N]} u_j$ with success probability $1 - \delta$ in time $O(T\sqrt{N}\log(\frac{1}{\delta}))$.

V. QUANTUM MATCHING PURSUIT

Now that we have introduced the necessary quantum background, we show how to construct the quantum matching-pursuit algorithm. The proposed algorithm closely follows its classical counterpart, but it takes advantage of the quantum routines presented in Sec. IV D. The main idea is to exploit the quantum regime to speed up the sweep stage.

A. Data access

First, we need quantum access to the dictionary D. We recall that, without loss of generality, we can consider the columns of D to be ℓ_2 normalized vectors, meaning that the Frobenius norm of D is $\|D\|_F = \sqrt{m}$. Recall that, with a preprocessing time of $O(nm\log^2(nm))$, we can create quantum access to the following quantum state in time $\widetilde{O}(1)$:

$$|D\rangle = \frac{1}{\sqrt{m}} \sum_j^m |j\rangle |d_j\rangle , \tag{20}$$

where d_j is the jth column of D and $|d_j\rangle$ is a state vector. This preprocessing cost needs to be paid only once and allows applying the quantum matching pursuit on any signal $s \in \mathbb{R}^n$ that is sparse over D. For this reason, we will not include this cost in our run-time analysis.

Similarly, we can create quantum access to the residual $r \in \mathbb{R}^n$ by using a tree data structure. For the sake of clarity, we report the data structure in Fig. 1. This data structure can be created in time $O(n\log(n))$ for a vector with n components and enables quantum access to the following quantum state in time $\widetilde{O}(1)$ [22,27]:

$$|r\rangle = \frac{1}{\|r\|_2} \sum_i^n r_i |i\rangle . \tag{21}$$

Preparing access to $|r\rangle$ and $|D\rangle$ means implementing the circuits described by the unitaries discussed in Sec. IV C.

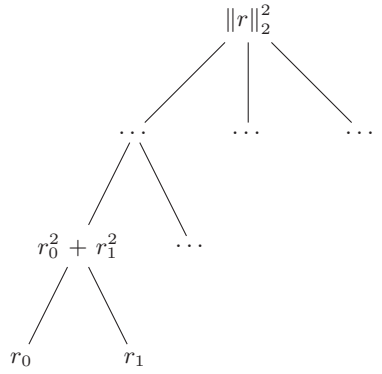

FIG. 1. The tree structure that enables efficient quantum access to the vector of the residuals. Each node stores the sum of squares of the leaves that descend from that node.

Finally, both the signal $s \in \mathbb{R}^n$ and its sparse representation $x \in \mathbb{R}^m$ are represented as classical arrays, as they will not be encoded in quantum states. To further lower the memory complexity, it is possible to exploit the sparseness of x and store it in a data structure which uses $O(\|x\|_0)$ space (e.g., a hash table). This will not affect the overall time complexity as long as the data structure has a constant time insertion or update cost.

B. Quantum algorithm

We start by initiating the residual structure with the signal components. Since we have quantum access to the residual and to the dictionary as in Eqs. (21) and (20), we can use the matrix-vector product estimation procedure from Corollary 1 to produce the state

$$|\varphi\rangle = \frac{1}{\sqrt{m}} \sum_j^m |j\rangle |\bar{z}_j\rangle . \tag{22}$$

We recall that $z_j = \langle d_j, r\rangle$ and that the procedure computes $\bar{z}_j$ such that

$$|\bar{z}_j - z_j| \leqslant \xi \|r\|_2, \tag{23}$$

where $\xi \in \mathbb{R}^+$ is a parameter of arbitrary choice and $\|r\|_2$ is the residual's ℓ_2 norm at the current iteration.

Once we have a quantum register $|\varphi\rangle$ with a superposition of all the inner products, we can perform the "finding the maximum absolute value" routine from Corollary 2 to find j^* and z_{j^*}. With the best j and z_j, we can proceed to update the solution $x \in \mathbb{R}^m$ and the residual $r \in R^n$.

Just like in the classical algorithm, we repeat this procedure until the norm of the residual is lower than a threshold ϵ or the solution is such that $\|x\|_0 > L$ for a threshold $L \in \mathbb{N}^+$.

Algorithm. 2 concisely summarizes the quantum matching-pursuit procedure.

It is important to remark that the error in the inner products can introduce convergence issues and could lead to worse solutions than the ones provided by the classical algorithm. We have identified two alternative versions of the quantum matching-pursuit algorithm.

(1) *Single error.* The error affects only the computation of j^*. We recompute z_{j^*} classically after the finding procedure outputs the atom's index.

Algorithm 2. Quantum matching pursuit.

1: Initialize $r = s$, $x = 0^{\otimes m}$.
2: **while** not ($\|x\|_0 > L$ or $\|r\|_2 \leqslant \epsilon$) **do**
3: Prepare $|r\rangle = \frac{1}{\|r\|_2}\sum_i^n r_i\,|i\rangle$
4: Prepare $|D\rangle = \frac{1}{\sqrt{m}}\sum_j^m |j\rangle\,|d_j\rangle$
5: Use inner product estimation to create $|\varphi\rangle = \frac{1}{\sqrt{m}}\sum_j^m |j\rangle\,|\bar{z}_j\rangle$, where $z_j = \langle d_j, r\rangle$ and $|\bar{z}_j - z_j| \leqslant \xi\|r\|_2$
6: Apply finding the maximum absolute value to $|\varphi\rangle$ to obtain j^* and z_{j^*}
7: Update the solution $x_{j^*} = x_{j^*} + z_{j^*}$
8: Update quantum access to the residual $r = r - z_{j^*}d_{j^*}$
9: **end while**
10: Output x.

(2) *Double error.* The error affects the computation of both j^* and z_{j^*}, which are retrieved quantumly by the finding procedure.

While the asymptotic complexity of the algorithm does not change between the two versions, the second one is slightly faster to compute. In Sec. VI A, we provide experimental evidence that the two versions of the algorithm do not significantly affect the performance of the quantum matching pursuit in practice, supporting the use of the second version. We also provide numerical experiments to check whether the quantum algorithm can find representations of the same quality as its classical counterpart.

C. Success probability

Different from its classical counterpart, the quantum algorithm that we propose has a probability of failing. In this section, we discuss the success probability of our algorithm, showing that we can arbitrarily bound its failure probability with little running-time overhead.

The probability of failure is due to the computation of the inner products at step 5 and the search at step 6. In particular, Corollaries 1 and 2 fail with a probability smaller than $2\delta_1'$ and δ_2', with run-time overheads of $O(\log(\frac{m}{\delta_1'}))$ and $O(\log(\frac{1}{\delta_2'}))$, respectively.

Let us denote the probability of failure of the ith loop iteration by $p_f(i)$ and consider $\delta_1' = \delta_2' = \delta'$. The probability of success of the ith loop iteration is

$$1 - p_f(i) \geqslant (1-2\delta')(1-\delta') = 1 - 3\delta' + 2\delta'^2 \geqslant 1 - 3\delta', \tag{24}$$

with a run-time overhead of $O(\log(\frac{m}{\delta'})\log(\frac{1}{\delta'})) = O(\log(\frac{m+1}{\delta'}))$. So the probability of failure of the ith loop iteration is bounded by $3\delta'$.

Given that the loop is executed k times, by union bound, we can bound the total probability of failure as

$$p\Big(\cup_i^k p(i)\Big) \leqslant \sum_i^k p(i) \leqslant 3k\delta'. \tag{25}$$

It follows that the success probability of our algorithm is $1 - 3k\delta'$ with a run-time overhead of $O(\log(\frac{m+1}{\delta'}))$.

By choosing $\delta' = \frac{\delta}{3k}$, we can affirm that our algorithm succeeds with probability greater than $1-\delta$ with a run-time overhead of $O(\log(\frac{3k(m+1)}{\delta})) \sim O(\log(\frac{3km}{\delta}))$.

Using the same proof technique, we could choose two different probabilities of failure δ_1' and δ_2' and further reduce the overhead by a constant factor. For instance, by choosing $\delta_1' = \frac{\delta}{2k}$ and $\delta_2' = \frac{\delta}{2km}$, we can bound the run-time overhead with $O(\frac{2km}{\delta})$. Finally, note that even if the exact value of k is not known beforehand, we can find suitable δ_1' and δ_2' by considering $k \sim O(L)$.

D. Running time

Finally, we provide a thorough analysis of the run time of the proposed algorithm, proving its computational complexity. As already stated, we assume that the dictionary has been stored in an appropriate data structure, and we analyze the time to compute the sparse representation of a signal over that dictionary.

The first step of the algorithm consists of initializing the residual. This can be done by building the data structure in Fig. 1, and it requires time $O(n\log(n))$. Since the residual and the dictionary are stored in adequate data structures, the preparation of states $|D\rangle$ and $|r\rangle$ at steps 3 and 4 is $\widetilde{O}(1)$, as discussed in Sec. IV C.

Given that the cost of preparing the two quantum states is $\widetilde{O}(1)$, from Corollary 1, we know that the cost of step 5 is $\widetilde{O}(\frac{1}{\xi})$ and that it encodes in the register values $\bar{z}_j$, with error $|\bar{z}_j - z_j| \leqslant \xi\|r\|_2$. Therefore, we can prepare the state

$$|\varphi\rangle = \frac{1}{\sqrt{m}}\sum_j^m |j\rangle\,|\bar{z}_j\rangle \tag{26}$$

in time $\widetilde{O}(\frac{1}{\xi})$.

We can ignore the run-time terms that depend on δ as they are related to the success probability studied in the previous section. We will include this overhead in the run time at the end of this section.

Recall that the state above is the superposition of m inner products, among which we need to identify the one with the greatest maximum absolute value. Using Corollary 2, we can find the value z_{j^*} and its index j^* in time $O(\frac{\sqrt{m}}{\xi})$.

Once these values are computed, we need to update the solution and the residual. To update the solution, we need to add $\bar{z}_{j_*}$ to one component of the solution vector. We can do it in time $O(1)$. On the other hand, updating the residual is more demanding, as we need to modify the tree data structure. Each element update costs $O(\log(n))$, as we need to update one leaf and all its parent nodes. Since the update is $r = r + z_{j^*}d_{j^*}$, we need to update $\|d_{j^*}\|_0$ elements of the residual. This step costs $O(\|d_{j^*}\|_0\log(n))$.

Let us denote by $d_{j^*}^{(i)}$ the best atom at iteration i. The cost of the ith iteration of the loop results in

$$\widetilde{O}\left(\|d_{j^*}^{(i)}\|_0\log(n) + \frac{\sqrt{m}}{\xi}\right). \tag{27}$$

By assuming that we perform k iterations of the loop, considering the initial cost of initiating the residual and introducing the term that accounts for the success probability, the total complexity of our algorithm is

$$\widetilde{O}\left(n\log n+\sum_{i}^{k}\left\|d_{j*}^{(i)}\right\|_0\log(n)+k\frac{\sqrt{m}}{\xi}\log\left(\frac{3km}{\delta}\right)\right). \quad (28)$$

If the atoms of our dictionary are not sparse, we can write the run time in a more compact way by observing that $\forall j$, $\|d_j\|_0 \leqslant n$,

$$\widetilde{O}\left(kn\log n+k\frac{\sqrt{m}}{\xi}\log\left(\frac{3km}{\delta}\right)\right). \quad (29)$$

We can concisely summarize our result in the following theorem.

Theorem 1. Quantum matching pursuit. Let there be quantum access to a dictionary $D \in \mathbb{R}^{n\times m}$ and to a vector $s \in \mathbb{R}^n$. Let $\xi, \delta \in \mathbb{R}_{>0}$ be precision parameters. There exists a quantum algorithm that simulates the matching pursuit algorithm, with error $|z_j^{(i)} - \overline{z}_j^{(i)}| \leqslant \xi\|r^{(i)}\|_2$ on the inner product estimation at each ith iteration, in running time $\widetilde{O}(kn\log n + k\frac{\sqrt{m}}{\xi}\log(\frac{3km}{\delta}))$, where k is the total number of iterations. The algorithm succeeds with probability greater than $1-\delta$.

VI. NUMERICAL EXPERIMENTS

The quantum algorithm that we propose executes the exact steps of its classical counterpart, but it introduces some random error along the computation, possibly affecting the quality of the representation. We present numerical simulations on synthetic data to better study the run-time advantage and the convergence properties of the quantum matching pursuit. We study whether the single-error and double-error versions of the algorithm, introduced in Sec. V B, actually affect the algorithm's correctness. Our numerical simulations are performed on a classical computer by introducing artificial errors $\xi \in [-0.01, 0.01]$ in the computation of the inner products.

A. Single- and double-error versions and representation quality

We run experiments to test whether the two implementations of the quantum matching pursuit algorithm produce different results and whether the representation quality is different from the one obtained by the classical matching pursuit.

To do so, we proceed by creating 100 batches, each of which composed of (1) a random dictionary $D \in \mathbb{R}^{100\times 512}$, with unit columns, (2) 100 sparse vectors $x \in \mathbb{R}^m$, with $\|x\|_0 = 17$, and (3) 100 signals of the form $s = Dx + \epsilon$, where ϵ is a vector containing Gaussian truncated noise. The dictionary, the signals, and the sparse vectors are provided by SCIKIT-LEARN's [28] MAKE_SPARSE_CODED_SIGNAL function. We add the ϵ noise artificially.

We simulate the two versions of the quantum matching-pursuit algorithm and the classical one for each batch to compare the sparse representations of the same data. In order to assess the convergence properties, we do not set any threshold L that limits the sparsity of the solution.

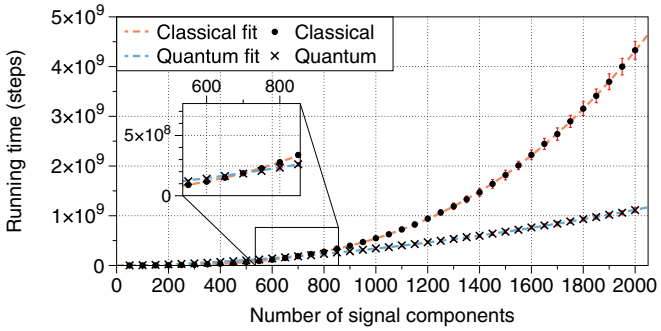

FIG. 2. Running times for classical and quantum matching pursuits. Each point is the average number of operations required to find the sparse representation of a signal $s \in \mathbb{R}^n$. We report the standard deviation with red bars.

We observe that all three versions represent the signals with 18 components on average. For both the single-error and the double-error versions, we compute, for each batch, the average sparsity of the solution for that batch, divided by the one obtained using the classical version. The average values for this metric are, respectively, 1.0048 and 1.0060, indicating that the quantum and classical algorithms compute solutions with similar sparsity and also that the two quantum versions do not differ much.

To better prove the latter point, we run a statistical test. First, we check whether the metric values obtained for each quantum algorithm version are normally distributed, using a Shapiro-Wilk normality test.

The resulting p values are, respectively, 0.37490 for the double-error version and 0.07854 for the single-error one. Since the single-error results are not normally distributed and the experiments were conducted on the same batches of data, we run a Wilcoxon signed-rank test to determine whether the performances are different. The test outcome is a p value of 0.34790, which does not allow us to reject the hypothesis that the two algorithms perform similarly.

B. Run-time simulation

While it is true that the classical complexity of the matching pursuit, in the general case, scales as $O(knm)$ and that the quantum version scales as $O(kn\log(n) + k\frac{\sqrt{m}}{\xi}\log(\frac{3km}{\delta})\text{polylog}(nm))$, it is legitimate to wonder whether the number of iterations k is the same for both the classical and quantum versions.

To analyze the run time of the algorithms, we compare the classical matching pursuit with the double-error quantum matching pursuit for the same data set.

We generate data sets in the same way as discussed in the previous section, with the difference that we study the run time as the length n of the signal increases. For each batch, the number of atoms m is set to twice the length of the signal, while the solution is five times sparser than the original signal.

Experimentally, we see that there is not a significant difference in the number of iterations required to converge. Indeed, the two algorithms produce solutions of the same sparsity. Figure 2 illustrates the run times of the classical and quantum algorithms, considering a probability of failure $\delta = 0.01$.

Each point represents the average number of operations required to find the sparse representation of the signal, while the red bars show the standard deviation.

We notice that the quantum algorithm is not advantageous until a certain number of signal components. While the exact signal length for quantum advantage depends on the characteristics of the problem (e.g., number of atoms, expected sparsity), this experiment shows that the quantum matching algorithm provides a substantial speedup over its classical counterpart for high-dimensional signals.

VII. CONCLUSIONS

In this work, we have presented a quantum algorithm to find sparse representations of signals in the QRAM input model. This algorithm is a quantum version of the classical matching-pursuit algorithm. We have leveraged, modified, and combined two quantum subroutines to speed up the sweep stage of the matching pursuit. The result is a routine with a polynomial advantage over its classical counterpart while retrieving solutions of the same quality.

This work shows that quantum computing can impact learning sparse representation and opens the path to further research on other quantum pursuit algorithms.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Alessandro Luongo for his valuable advice and Prof. Ferruccio Resta and Prof. Donatella Sciuto for their support. The authors are particularly grateful to Prof. Giacomo Boracchi for his inspiring lectures on sparse representations and to Prof. Timothy J. Sluckin for his precious feedback on the first draft of this paper.

[1] H. Rauhut, K. Schnass, and P. Vandergheynst, Compressed sensing and redundant dictionaries, IEEE Trans. Inf. Theory **54**, 2210 (2008).

[2] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still Image Data Compression Standard* (Kluwer Academic Publishers, Assinippi Park, Norwell, MA, 1992).

[3] M. Elad and M. Aharon, Image denoising via sparse and redundant representations over learned dictionaries, IEEE Trans. Image Process. **15**, 3736 (2006).

[4] A. Adler, M. Elad, Y. Hel-Or, and E. Rivlin, Sparse coding with anomaly detection, J. Signal Process. Syst. **79**, 179 (2015).

[5] A. Bellante, A. Luongo, and S. Zanero, Quantum algorithms for data representation and analysis, arXiv:2104.08987.

[6] I. Kerenidis and A. Luongo, Classification of the MNIST data set with quantum slow feature analysis, Phys. Rev. A **101**, 062327 (2020).

[7] I. Kerenidis and J. Landman, Quantum spectral clustering, Phys. Rev. A **103**, 042415 (2021).

[8] S. G. Mallat and Z. Zhang, Matching pursuits with time-frequency dictionaries, IEEE Trans. Signal Process. **41**, 3397 (1993).

[9] S. Krstulovic and R. Gribonval, Mptk: Matching pursuit made tractable, in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings* (IEEE, Piscataway, NJ, 2006), Vol. 3, pp. III-496–III-499.

[10] Y. Wu and V. S. Batista, Matching-pursuit for simulations of quantum processes, J. Chem. Phys. **118**, 6720 (2003).

[11] Y. Wu and V. S. Batista, Quantum tunneling dynamics in multidimensional systems: A matching-pursuit description, J. Chem. Phys. **121**, 1676 (2004).

[12] X. Chen and V. S. Batista, Matching-pursuit/split-operator-Fourier-transform simulations of excited-state nonadiabatic quantum dynamics in pyrazine, J. Chem. Phys. **125**, 124313 (2006).

[13] Y. Wu, M. F. Herman, and V. S. Batista, Matching-pursuit/ split-operator Fourier-transform simulations of nonadiabatic quantum dynamics, J. Chem. Phys. **122**, 114114 (2005).

[14] B. K. Natarajan, Sparse approximate solutions to linear systems, SIAM J. Comput. **24**, 227 (1995).

[15] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, Strengths and weaknesses of quantum computing, SIAM J. Comput. **26**, 1510 (1997).

[16] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2010).

[17] V. Giovannetti, S. Lloyd, and L. Maccone, Architectures for a quantum random access memory, Phys. Rev. A **78**, 052310 (2008).

[18] S. Arunachalam, V. Gheorghiu, T. Jochym-O'Connor, M. Mosca, and P. V. Srinivasan, On the robustness of bucket brigade quantum ram, New J. Phys. **17**, 123010 (2015).

[19] C. T. Hann, G. Lee, S. M. Girvin, and L. Jiang, Resilience of quantum random access memory to generic noise, PRX Quantum **2**, 020311 (2021).

[20] A. Paler, O. Oumarou, and R. Basmadjian, Parallelizing the queries in a bucket-brigade quantum random access memory, Phys. Rev. A **102**, 032608 (2020).

[21] P. Rebentrost, M. Santha, and S. Yang, Quantum alphatron, arXiv:2108.11670.

[22] I. Kerenidis and A. Prakash, Quantum recommendation systems, in *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)* (Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017), pp. 49-1–49-21.

[23] I. Kerenidis, J. Landman, A. Luongo, and A. Prakash, q-means: A quantum algorithm for unsupervised machine learning, in *Advances in Neural Information Processing Systems*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc., 2019), Vol. 32.

[24] A. Ta-Shma, Inverting well conditioned matrices in quantum logspace, in *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing* (Association for Computing Machinery, New York, 2013), pp. 881–890.

[25] C. Durr and P. Hoyer, A quantum algorithm for finding the minimum, arXiv:quant-ph/9607014.

[26] L. K. Grover, A fast quantum mechanical algorithm for database search, in *Proceedings of the Twenty-Eighth Annual*

ACM Symposium on Theory of Computing (Association for Computing Machinery, New York, 1996), pp. 212–219.

[27] L. Grover and T. Rudolph, Creating superpositions that correspond to efficiently integrable probability distributions, arXiv:quant-ph/0208112.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. **12**, 2825 (2011).