Solving Systems of Linear Equations: HHL from a Tensor Networks Perspective

Alejandro Mata Ali*
i3B Ibermatica, Quantum Development Department, Paseo Mikeletegi 5, 20009 Donostia, Spain

Iñigo Perez Delgado† and Marina Ristol Roura‡
i3B Ibermatica, Parque Tecnológico de Bizkaia, Ibaizabal Bidea, Edif. 501-A, 48160 Derio, Spain

Aitor Moreno Fdez. de Leceta§
i3B Ibermatica, Unidad de Inteligencia Artificial, Avenida de los Huetos, Edificio Azucarera, 01010 Vitoria, Spain

Sebastián V. Romero¶
TECNALIA, Basque Research and Technology Alliance (BRTA), 48160 Derio, Spain
(Dated: June 29, 2025)

We present a new approach for solving systems of linear equations with tensor networks based on the quantum HHL algorithm. We first develop a novel HHL in the qudits formalism, the generalization of qubits, and then transform its operations into an equivalent classical HHL, taking advantage of the non-unitary operations that they can apply. The main novelty of this proposal is to perform a classical simulation as efficiently as possible of the HHL to benchmark the algorithm steps according to its input parameters and the input matrix. We apply this algorithm to three simulation problems, comparing it with an exact inversion algorithm, and we compare its performance against an implementation of the original HHL simulated in the Qiskit framework, providing both codes. Our results show that our approach can achieve a promising performance in computational efficiency to simulate HHL process without quantum noise, providing a lower bound.

Keywords: Tensor networks, Systems of linear equations, Quantum-inspired

I. INTRODUCTION

The solution of linear equation systems $A\vec{x} = \vec{b}$ is a fundamental problem in many areas of science and engineering. Classical methods for solving these equations, such as *Gaussian elimination* and *LU decomposition* [1], have been widely used and optimized for decades [2, 3]. However, as the size of the system grows, classical methods become computationally expensive and inefficient. One of the most efficient classical methods is the *conjugate gradient method* (CG) [4, 5], which has a complexity of $O\left(Ns\kappa \log\left(\frac{1}{\epsilon}\right)\right)$ for a matrix $N \times N$ with a maximum of s non-zero elements per row, $\kappa \equiv \frac{\lambda_{max}}{\lambda_{min}}$, being λ the eigenvalues of the matrix A and ϵ the error.

Quantum computers offer the potential to solve some challenging problems more efficiently than classical computers. In particular, the *HHL algorithm* proposed by Harrow, Hassidim, and Lloyd in 2008 [6, 7] is a method for solving linear equations that runs in polynomial time, where the polynomial depends logarithmically on the size of the system. However, it is intended for the calculation of the expectation values in $O\left(\log(N)s^2\kappa^2/\epsilon\right)$, since it loses its advantage in the case of extracting the explicit solution.

Recently, there has been growing interest in using qudits [8] and tensor networks [9, 10] to implement different quantum algorithms. Qudits are generalized qubits with more than 2 basis states. Tensor networks are classical representations of tensor algebra equations, providing an efficient way to represent and manipulate certain types of high-dimensional systems, such as quantum states with low entanglement [11], or compressing machine learning models [12, 13], enabling quick computations with classical computers [14, 15].

In this paper, we propose a novel approach for solving linear equations using qudits and tensor networks. We demonstrate how this approach can be used to efficiently solve systems of linear equations with a large number of variables, and we compare the performance of our approach with existing quantum and classical methods. The main contributions of this work are the following.

- The formulation of a qudit HHL algorithm.
- The formulation and implementation of a classical tensor network algorithm to classically simulate the best expected performance of the HHL algorithm.
- Comparison of this algorithm against the conjugate gradient method in different scenarios.

Our results show that our approach can achieve a promising performance in computational efficiency to simulate the HHL process without quantum noise and solving systems of linear equations. However, it does not improve the performance of the state-of-the-art best-

* alejandro.mata.ali@gmail.com
† iperezde@ayesa.com
‡ mristol@ayesa.com
§ aitormoreno@lksnext.com
¶ sebastian.vidal@tecnalia.com

known linear solver algorithm, and it is not intended to do so.

The paper is organized as follows. Initially, in Sec. II, we present a concise overview of classical algorithms designed to solve linear equations, as well as the HHL algorithm, omitting detailed quantum computing concepts. Subsequently, in Sec. III, we introduce an innovative qudit algorithm that enhances the original HHL. Following this, Sec. IV describes the classical tensor network approach to simulate the optimal HHL performance and analyze its complexity. Sec. V is devoted to comparing our novel algorithm with the conjugate gradient method and traditional HHL. In the final section, Sec. VI, we apply the algorithm to three simulation problems, contrasting it with an exact inversion algorithm, and evaluate its performance relative to an implementation of the HHL algorithm simulated via the Qiskit framework. All code is available in the GitHub repository https://github.com/DOKOS-TAYOS/Tensor_Networks_HHL_algorithm.git and there is a brief spanish explanation of the paper in the video Algoritmo HHL con Tensor Networks y qudits.

II. BACKGROUND

There exist several algorithms for solving systems of linear equations, but we will introduce only a few interesting ones. All of them solve the system of linear equations

$$A\vec{x} = \vec{b}, \tag{1}$$

where A is an invertible matrix $N \times N$, $\vec{x}$ is the vector we want to obtain and $\vec{b}$ is another vector, both of dimension N.

The first is *Gaussian elimination*, which consists of reducing the augmented matrix $[A|b]$ to a matrix of the row echelon through row addition operations and transforming it into a diagonal matrix with the required solutions. Its computational complexity is $O(N^3)$. It is widely used in small cases, but when matrices become too large, other methods are applied because of its prohibitive cubic cost. Another extended algorithm is the *LU decomposition* [1], which can be viewed as another way to perform Gaussian elimination. It decomposes the matrix A into two matrices L (lower triangular) and U (upper triangular) that satisfy $A = LU$. It has the same complexity as Gaussian elimination. Following the same line, there is the *Cholesky decomposition* [16], which consists of decomposing a Hermitian positive definite matrix A into a lower triangular matrix L with real positive diagonal entries that satisfy $A = LL^*$. Its complexity is $O(n^3)$, but it has half the cost of LU decomposition. This method is widely used for Monte Carlo simulation.

In the field of iterative methods, the most simple one is the *Jacobi algorithm* [17]. First, we decompose the matrix A into three matrices D diagonal, L lower triangular, and U upper triangular such that $A = D + L + U$. Then, we apply for the k-th iteration

$$\vec{x}^{(k+1)} = D^{-1}\left(\vec{b} - (L+U)\vec{x}^{(k)}\right) \tag{2}$$

until it converges. Its complexity is $O(N^2)$ for each step in the dense case and can take advantage of the sparsity of A. For this reason, it is used in large problems.

An improvement of this method is the *Gauss-Seidel* algorithm for definite positive matrices, or strictly diagonally dominant matrices, with no zero diagonal elements. In this case, the iteration is made with

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right). \tag{3}$$

This method has the advantage in the storage, allowing one to rewrite the first iteration vector, which enables to address larger problems, but it is much harder to implement in parallel.

Another iterative algorithm is the well-known *conjugate gradient method* [4] for large sparse positive semidefinite matrices. Two non-zero vectors $\vec{v}$ and $\vec{w}$ are conjugate with respect to A if

$$\vec{v}^T A\vec{w} = 0. \tag{4}$$

If we define a set $P = \{\vec{p}_0, \vec{p}_1, \ldots, \vec{p}_{N-1}\}$ of mutually conjugate vectors with respect to A, then P forms a base on $\mathbb{R}^N$ and

$$\vec{x} = \sum_{i=0}^{N-1} \alpha_i \vec{p}_i \implies A\vec{x} = \sum_{i=0}^{N-1} \alpha_i A\vec{p}_i. \tag{5}$$

Multiplying the problem by the left with $\vec{p}_j^T$

$$\vec{p}_j^T \cdot \vec{b} = \vec{p}_j^T A\vec{x} = \sum_{i=0}^{N-1} \alpha_i \vec{p}_j^T A\vec{p}_i = \alpha_j \vec{p}_j^T A\vec{p}_j, \tag{6}$$

then

$$\alpha_j = \frac{\vec{p}_j^T \cdot \vec{b}}{\vec{p}_j^T A\vec{p}_j}. \tag{7}$$

So, we need to obtain a set P and then compute the α_j values. However, if we choose good $\vec{p}_i$ vectors, we may not need to compute all of them to solve the problem. So, we will solve the problem approximately. We have an initial guess $\vec{x}_0$, and we define a function to minimize

$$f(\vec{x}) = \frac{1}{2}\vec{x}^T A\vec{x} - \vec{x}^T \cdot \vec{b}. \tag{8}$$

Due to $\nabla f(\vec{x}) = A\vec{x} - \vec{b}$, we take $\vec{p}_0 = \vec{b} - A\vec{x}_0$. Being $\vec{r}_j$ the residual of the j-th step, and the negative gradient of f in $\vec{x}_j$

$$\vec{r}_j = \vec{b} - A\vec{x}_j. \tag{9}$$

To obtain a vector $\vec{p}_j$ conjugate to the other ones, we apply

$$\vec{p}_j = \vec{r}_j - \sum_{i<j} \frac{\vec{r}_j^T A \vec{p}_i}{\vec{p}_i^T A \vec{p}_i} \vec{p}_i, \tag{10}$$

and we update $\vec{x}_{j+1} = \vec{x}_j + \alpha_j \vec{p}_j$ with

$$\alpha_j = \frac{\vec{p}_j^T \cdot \left(\vec{b} - A\vec{x}_j\right)}{\vec{p}_j^T A \vec{p}_j} = \frac{\vec{p}_j^T \cdot \vec{r}_j}{\vec{p}_j^T A \vec{p}_j} \tag{11}$$

until the residual is small enough.

Similar methods such as the *generalized minimal residual method* [18] and the *biconjugate gradient stabilized method* [19] are based on similar ideas as improvements for more general problems. All of these algorithms require matrix-vector multiplications, so their complexity is at least linear in the dimension of the problem.

In this context, the *quantum linear solver algorithm* HHL [6] provides a computational advantage. Now, we briefly introduce the standard HHL algorithm in qubits in order to better understand the algorithm we will formalize.

For this algorithm, we need $n = \lceil \log_2 N \rceil$ qubits to encode the vector $\vec{b}$, n_c clock qubits to encode the possible eigenvalues of A and one auxiliary qubit for the inversion with respect to these eigenvalues. The whole circuit can be summarized in Fig. 1.

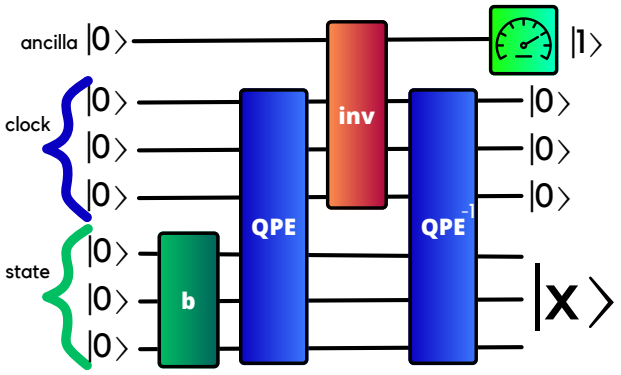

FIG. 1. Quantum HHL in qubits with $n = n_c = 3$.

We encode the state $\vec{b}$ in the amplitudes of the n state qubits

$$|b\rangle = \sum_{i=0}^{2^n-1} b_i |i\rangle = \sum_{i=0}^{2^n-1} \beta_i |u_i\rangle, \tag{12}$$

where b_i are the normalized components of the vector $\vec{b}$, $|i\rangle$ are the computational bases states and $|u_i\rangle$ the eigenvector associated with the eigenvalue λ_i of A. We use an operator b to initialize it. It is important to note that the difference between N and 2^n will be completed with zeros in the vector and a matrix proportional to the identity in A, wasting resources. Moreover, we need a method to generate this state $|b\rangle$ or the b operator [20].

The second thing we do is calculate the evolution operator

$$U = e^{iA'\tau}, \tag{13}$$

being τ a hyperparameter to rescale the eigenvalues for the next step and the matrix $A' = A$ if A is Hermitian, and

$$A' = \begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix} \tag{14}$$

if it is not. In this case, the problem is

$$A' \begin{pmatrix} 0 \\ \vec{x} \end{pmatrix} = \begin{pmatrix} \vec{b} \\ 0 \end{pmatrix}. \tag{15}$$

We assume that U can be calculated and implemented efficiently [21].

With this operator U presented, we perform a *Quantum Phase Estimation* (QPE) [22] to encode the eigenvalues of A' in the clock qubits. Since we have n_c qubits, we can encode the $\mu = 2^{n_c}$ possible values of eigenvalues. For this reason, n_c (and therefore μ) is a hyperparameter for resolution, since we will have to choose it based on the properties of the problem, usually the conditional number of the matrix.

Now we apply an inversion operator, which rotates the probability of the auxiliary qubit so that it is divided by the value of the eigenvalue encoded by the QPE.

The next step is to make a post-selection, keeping only the state if the auxiliary qubit outputs a $|1\rangle$, followed by an inverse QPE to clean the eigenvalue qubits.

In the end, we have the $\vec{x}$-state normalized in the amplitudes,

$$|x\rangle = \frac{1}{\mathcal{N}} \sum_{i=0}^{2^n-1} \frac{\beta_i}{\lambda_i} |u_i\rangle = \frac{1}{\mathcal{N}} \sum_{i=0}^{2^n-1} x_i |i\rangle, \tag{16}$$

with a normalization constant $\mathcal{N}$ and omitting the ancilla and clock qubits.

To obtain the full state vector, we first have to obtain the probability distribution, which requires measuring so many times that the computational advantage is lost. For this reason, the HHL is intended to obtain the expectation value of some operator with respect to the solution, or to introduce the solution state into another quantum subroutine.

The main problems of the algorithm are:

1. Large amount and waste of resources due to the difference between the size N of the problem and the n qubits to encode it.

2. Circuit depth and errors introduced by the state preparation and the QPE, in addition to the need for probabilistic post-selection.

3. We do not get $\vec{x}$ directly, and if we extract it, we get it with a sign ambiguity for each of its elements.

4. The preparation of state $\vec{b}$ may not be trivial, just like performing the inversion operator or performing the U operator.

III. QUDIT QUANTUM ALGORITHM

To try to overcome the first two problems of the HHL, we formalize a qudit version of the algorithm. To do so, we assume that there are quantum computers that implement the basic qudit gates as described in the paper [8]. This assumption is basic, since our work is only theoretical, without going into all the technical difficulties of the hardware. This section is simply a mathematical motivation for the last tensor network algorithm.

The first thing we do is encode the state $\vec{b}$ in a single qudit. In case the qudit does not have enough states available, we encode it in a number of qudits that allows us to encode it in a way analogous to the case of qubits. In the following, we assume that we only need one qudit with N basis states in order to clearly explain the algorithm.

It is important to understand that the goal of this encoding is to reduce the physical amount of quantum resources. We have to use a Hilbert space exponentially larger than each of the Hilbert spaces of dimension 2 that would be used in the original case. However, we have to consider that there are physical systems that have this number of states available, since this is the basic premise of the work. An example would be to have a particle with a quantum number with possible values $-3/2, -1/2, 1/2, 3/2$ instead of another particle with possible values $-1/2, 1/2$.

Moreover, this change in the space used for the encoding will not imply a loss of the computational advantage of the original algorithm in the case of being able to realize the gates expressed in the paper [8], since the unitary gate U is applied over the entire set of state qubits, as if it were a matrix over an ordinary vector.

Now we need a way to simulate the $U = e^{iA'\tau}$ operator, which depends on the particular case we are dealing with.

With this, we make the following circuit in Fig. 2.

With a single qudit of dimension $\mu = 2^{n_c}$ we can do the QPE as in [8] and encode the μ possible values of the eigenvalue in its basis states. However, we could use more qudits. If we use 2 qudits to encode μ values, each one will need to have dimension $\sqrt{\mu}$. This change in resources from clock qubits to a clock qudit will not imply a loss of quantum advantage either, since we will have the same number of possible eigenvalues.

The inverter is exactly the same as in the case of qubits, but instead of having a control-noncontrol series, we have a control i that applies the rotation gate to the ancilla if we have the value i in the qudit.

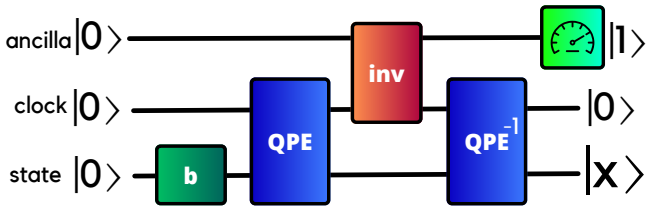

FIG. 2. Quantum HHL in qudits.

We do the post-selection and if we get $|1\rangle$, we perform the inverse QPE to clear the qudit of the eigenvalues.

With this, we can see that we reduce the number of SWAP gates needed and the QPE is performed with a low number of gates. Also, we waste less resources, as we can better adjust the dimensionality of the quantum system with respect to the equation to be solved, without the need to have extra elements at 0 to complete the 2^n components, requiring an extra qubit. Moreover, in the best-case scenario we only need two qudits and one qubit, and in the worst-case scenario we need the same resources as in the original HHL.

Still, we could do more to solve the other problems, so let us try to tackle them with the quantum-inspired technique of tensor networks, avoiding gate errors from the quantum devices and extracting the $\vec{x}$.

IV. TENSOR NETWORKS ALGORITHM

We will transform the qudit circuit into a tensor network, so that it returns the vector $\vec{x}$ directly. This algorithm is called the *tensor network HHL* (TN HHL). Since in tensor networks we do not need normalization, we will not normalize the state $|b\rangle$. As it is not a unit vector, the result state is not normalized either, so we do not have to rescale it. Moreover, the state can be prepared exactly in a single operation, defining the node $\vec{b}$ with dimension N.

The QPE is performed by contracting the uniform superposition clock state with the *Quantum Fourier Transform* (QFT) [23] in the QPE, so we have a matrix $H[\mu]$ with dimension $\mu \times \mu$ for μ eigenvalues with elements

$$H[\mu]_{ab} = e^{2\pi i \frac{ab}{c}}, \tag{17}$$

without normalization.

The inverter is a non-unitary operator with dimension $\mu \times \mu$ for μ eigenvalues whose non-zero elements are

$$\text{inv}[\mu]_{i,j} = \begin{cases} 1/i & \text{if } i = j \neq 0 \text{ and if } i \leq \frac{\mu}{2} \\ \frac{1}{i-\mu} & \text{if } i = j \text{ and if } i > \frac{\mu}{2} \end{cases} \tag{18}$$

in order to encode negative eigenvalues due to the cyclic property of the imaginary exponential. If we want more positive or more negative eigenvalues, we must change the proportion of i values that are translated as positive or negative eigenvalues.

The phase kickback operators can also be obtained exactly from U. This tensor P with dimension $N \times \mu \times N$ is

$$P[\mu]_{i,j,k} = \left(U^j\right)_{i,k}; \left(P[\mu]^{-1}\right)_{i,j,k} = \left(\left(U^{-1}\right)^j\right)_{k,i}. \quad (19)$$

These tensors are contracted through their j index with the $H[\mu]$ and $H[\mu]^{-1}$ tensors to perform QPE.

With these tensors, we can get our result by contraction of the tensor network in Fig. 3 a)

$$\sum_{a,b,c,d,e,f} b_a P_{abc} H^{-1}_{bd} \mathrm{inv}_{de} H_{ef} P^{-1}_{cfi} = \frac{x_i \mu^2}{\tau}, \quad (20)$$

with $b, c, d, e, f \in [0, \mu-1]$ and $a, c, i \in [0, N-1]$, omitting $[\mu]$, since it is the same for all the tensors.

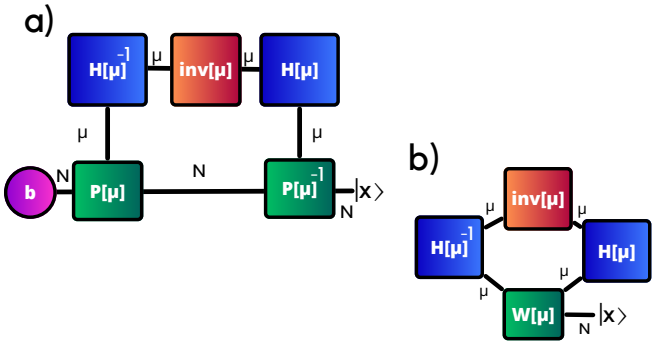

FIG. 3. Tensor network equivalent to HHL. a) Original way. b) Eficient way.

We assume that contracting a tensor of N indexes of dimension n_i with another of M indexes of dimension m_j through its first index has the usual computational cost of $O\left(n_0 \prod_{i,j=1,1}^{N-1,M-1} n_i m_j\right)$.

The computational complexity of contracting the tensors $H[\mu]^{-1}$ and $\mathrm{inv}[\mu]$ is $O\left(\mu^2\right)$, due to the sparsity of the tensor $\mathrm{inv}[\mu]$. Contracting the resulting tensor with $H[\mu]$ has a complexity of $O\left(\mu^3\right)$. The contraction of $\vec{b}$ with $P[\mu]$ has a complexity of $O\left(\mu N^2\right)$. The contraction of the tensors resulting from the contraction of $H[\mu]^{-1}$, $H[\mu]$ and $\mathrm{inv}[\mu]$ and the contraction of $\vec{b}$ and $P[\mu]$ has a computational complexity of $O\left(\mu^2 N\right)$. And the final contraction with $P[\mu]^{-1}$ has a complexity of $O\left(\mu N^2\right)$.

Therefore, the contraction of the tensor network has a computational complexity of $O\left(\mu \max\left(\mu^2, \mu N, N^2\right)\right)$.

The construction of the tensors has a complexity:

- $H[\mu]^{-1}$ and $H[\mu]$: $O\left(\mu^2\right)$.
- $\mathrm{inv}[\mu]$: $O\left(\mu\right)$.
- $\vec{b}$: $O\left(N\right)$.
- $P[\mu]$ and $P[\mu]^{-1}$: $O\left(\mu N^3\right)$, since we have to multiply up to μ times $N \times N$ matrices.

Therefore, the computational complexity of the algorithm is $O\left(\mu \max\left(\mu^2, \mu N, N^3\right)\right)$.

The complexity of calculating U depends on the sparsity of the A' matrix you use and the method. In the case of having n non-zero elements per column, the complexity of this multiplication is $O(nN^2 \log(\omega))$, with ω being a number related to the convergence of the method used. However, this is highly dependent on the matrices used. For example, for a tridiagonal matrix, we can do it in $O(N^2)$. Because of the complexity of adding this term to the complexities we are working with, we will omit it. The calculation of U^{-1} requires the same complexity.

We can avoid this increase in complexity by building the tensors $P[\mu]$ and $P[\mu]^{-1}$ by defining a tensor $W[\mu]$ that does the same as the contraction of both tensors with the vector $\vec{b}$. Its elements are

$$W[\mu]_{ijk} = \left(\vec{b} U^{i-j}\right)_k. \quad (21)$$

This tensor has dimension $\mu \times \mu \times N$, and its construction requires multiplying up to m times a vector with a matrix, which can be U or U^{-1}. To construct each element, we can take advantage of another previously calculated element. Instead of having to calculate U^{i-j}, we multiply U by the previously calculated vector $\vec{b}U^{i-j-1}$ or multiply U^{-1} by $\vec{b}U^{i-j+1}$. Therefore, the complexity of creating the tensor is $O\left(\mu N^2\right)$.

So, we have to contract the tensor network in Fig. 3 b)

$$\sum_{a,b,c,d} W_{abi} H^{-1}_{ac} \mathrm{inv}_{cd} H_{db} = \frac{x_i \mu^2}{\tau}. \quad (22)$$

The complexity of contracting tensors $H[\mu]^{-1}$, $H[\mu]$, and $\mathrm{inv}[\mu]$ is the same as before. Contracting this resulting tensor with $W[\mu]$ has a complexity of $O\left(\mu^2 N\right)$. Therefore, the total computational complexity is $O\left(\mu \max\left(\mu^2, \mu N, N^2\right)\right)$. However, if we precalculate the contraction of $H[\mu]$, $H[\mu]^{-1}$ and $inv[\mu]$ to use it every time, we could avoid the $O(\mu^3)$ term.

The memory cost is $O(N \max(\mu^2, N))$, being the first term associated with the tensor $W[\mu]$ and the second term associated with the matrix U.

We can also compute the inverse of A' just by erasing the b node from Fig. 3 a) and performing the contraction, at a cost $O\left(\mu \max\left(\mu^2, \mu N^2, N^3\right)\right)$.

V. COMPARISON OF ADVANTAGES AND DISADVANTAGES

We will compare the advantages and disadvantages of this algorithm in tensor networks against the conjugate gradient and quantum HHL in Table I. We assume

$\mu = O(\kappa)$. μ also depends on the error bounds we want. The higher μ implies lower error bounds if we properly adjust the τ hyperparameter. However, we are not able to determine this scaling, but it should have a scaling similar to that in the original HHL. It could be studied in future work.

Notice that we have not made use of the properties of the sparse matrices, as in the CG or the original HHL.

A. TN HHL vs CG

We can see that our algorithm is slower than the CG. However, TN HHL can invert the matrix A. Moreover, both algorithms benefit from efficient matrix product algorithms. Also, if we have the eigenvalues, in the TN HHL we can change $H[\mu]$ and $W[\mu]$ to use less resources by using $\mu = n_\lambda$, being n_λ the number of eigenvalues.

B. TN HHL vs HHL

The advantages of this TN HHL method over traditional quantum HHL are as follows:

1. We do not waste resources, the QFT is a simple matrix, and we do not need SWAP gates.
2. We can directly obtain the solution or some of its elements, and we can get the right signs from our solutions. In addition, we can obtain the inverse matrix.
3. We do not have to generate a complicated circuit for the initialization of the state and the time evolution of the phase kickback. We also do not have the errors introduced by the quantum gates.
4. We do not need the post-selection.
5. We do not use quantum resources.

However, when it comes to computing the expectation value, it is indeed significantly less efficient in complexity.

VI. EXPERIMENTS

We will test the effectiveness of the method by performing numerical simulations. We will solve the forced harmonic oscillator, the forced damped oscillator, and the 2D static heat equation with sources. Then, we will compare the performance of the TN HHL against our Qiskit implementation of the original HHL for random sparse matrices.

It is important to note that we are going to compare the resolution with respect to the system of linear equations, not with respect to the original differential equation itself. This is because this method solves the discretization we have made of the problem, so errors due to the discretization itself should not be taken into account. In each experiment, the hyperparameters are tuned by hand.

All experiments are performed in CPU, with an Intel(R) Core(TM) i7-14700HX 2.10 GHz and 16 GB RAM.

A. Forced harmonic oscillator

The differential equation we want to solve is

$$\frac{d^2x}{dt^2} + \frac{k}{m}x = F(t) \qquad (23)$$
$$x_0 = x(t=0); \qquad x_T = x(t=T)$$

where $F(t)$ is the external force dependent on time. For the experiments, we use a force $F(t) = C\sin(\nu t)$.

We use discretization with n time steps, i.e. $\Omega = -2 + \frac{k}{m}(\Delta t)^2$ and $F_j = (\Delta t)^2 C\sin(\nu j \Delta t)$.

$$\begin{pmatrix} \Omega & 1 & 0 & \cdots & 0 & 0 \\ 1 & \Omega & 1 & \cdots & 0 & 0 \\ 0 & 1 & \Omega & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & \Omega \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} F_1 - x_0 \\ F_2 \\ F_3 \\ \vdots \\ F_n - x_T \end{pmatrix} \qquad (24)$$

The result of inverting this system gives us the result in Fig. 4. As hyperparameters of the algorithm of the method, we use $\mu = 2000$ and $\tau = 6000$.

The root mean square error of our tensor network from the exact inversion was 6.4×10^{-6} and took 350 ms to run, compared to 249 μs of the exact inversion method of PyTorch.

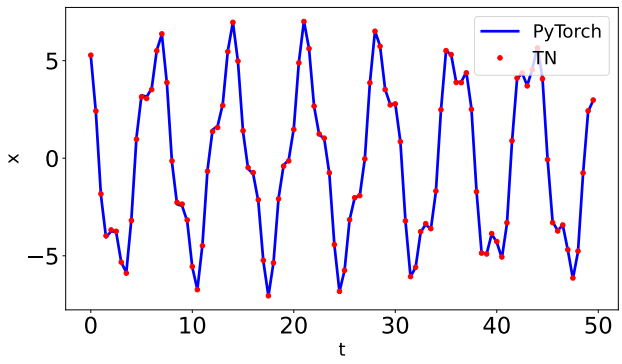

FIG. 4. Solving the forced harmonic oscillator system with equation (24). In blue the inversion performed with PyTorch and in red the inversion performed with the tensor network.

Algorithm	**Inversion** A^{-1}	**Solution** $\vec{x} = A^{-1}\vec{b}$	**Expectation value** $\vec{x^T} M \vec{x}$
CG	-	$O(Ns\kappa \log(1/\epsilon))$	$O(N \max(s\kappa \log(1/\epsilon), s'))$
HHL	-	-	$O(\log(N) s^2 \kappa^2/\epsilon)$
TN HHL	$O(N^2\kappa^2) + O(N^3\kappa) + O(\kappa^3)$	$O(N\kappa^2) + O(N^2\kappa) + O(\kappa^3)$	$O(\max(N^2\kappa, N\kappa^2, \kappa^3, Ns'))$

TABLE I. Computational times to invert a $N \times N$ matrix A, obtain the solution of $A\vec{x} = \vec{b}$ and compute an expectation value $\langle x|M|x\rangle$. s is the maximum number of non-zero elements per row of A and s' is the maximum number of non-zero elements per row of M.

B. Forced damped oscillator

The differential equation we want to solve is

$$\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} + \frac{k}{m}x = F(t) \qquad (25)$$
$$x_0 = x(t=0); \qquad x_T = x(t=T)$$

where $F(t)$ is the external force dependent on time and γ is the damp coefficient. As in Ssec. VI A, for the experiments we use a force $F(t) = C\sin(\nu t)$.

We use discretization with n time steps

$$\begin{pmatrix} \beta_0 & \beta_+ & 0 & \cdots & 0 & 0 \\ \beta_- & \beta_0 & \beta_+ & \cdots & 0 & 0 \\ 0 & \beta_- & \beta_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \beta_- & \beta_0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} F_1 - \beta_- x_0 \\ F_2 \\ F_3 \\ \vdots \\ F_n - \beta_+ x_T \end{pmatrix} \qquad (26)$$

where $\beta_- = 1 - \gamma\frac{\Delta t}{2}$, $\beta_+ = 1 + \gamma\frac{\Delta t}{2}$ and $\beta_0 = -2 + \frac{k}{m}(\Delta t)^2$. This matrix is not hermitian, so we apply (14) and solve (15).

The result of inverting this matrix gives us the result in Fig. 5. As hyperparameters of the algorithm of the method, we use $\mu = 2000$ and $\tau = 1.1 \times 10^4$.

The relative root mean square error of our tensor network from the exact inversion was 5.7×10^{-3} and took 1.26 seconds to run, compared to 183 μs of the Tensorflow method.

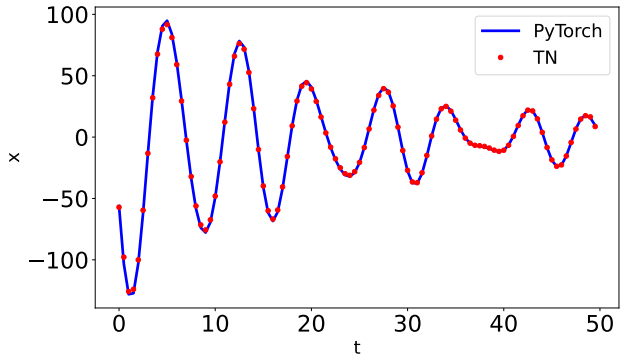

FIG. 5. Solving the forced damped oscillator system with equation (26). In blue the inversion performed with PyTorch and in red the inversion performed with the tensor network.

C. Static two dimensional heat equation with sources

The differential equation we want to solve is

$$k\left(\frac{d^2u}{dx^2} + \frac{d^2u}{dy^2}\right) = -S(x,y) \qquad (27)$$
$$u_{x1} = u(0,y); \quad u_{x2} = u(L_x,y)$$
$$u_y = u(x,0); \quad u_{y2} = u(x,L_y)$$

where $S(x,y)$ is the external source dependent on position. For the experiments, we used a source $S(x,y) = 10\sin\left(2\pi\frac{xy}{L_xL_y}\right)$ and $k = 1$.

We use discretization

$$u_{j+1,k} + u_{j-1,k} + u_{j,k+1} + u_{j,k-1} - 4u_{jk} = -(\Delta x)^2 S_{jk} \qquad (28)$$

We convert the 2-dimensional space into a line, create the matrix and obtain the following result in Figs. 6 and 7. As hyperparameters of the algorithm, we use $\mu = 2000$ and $\tau = 100$.

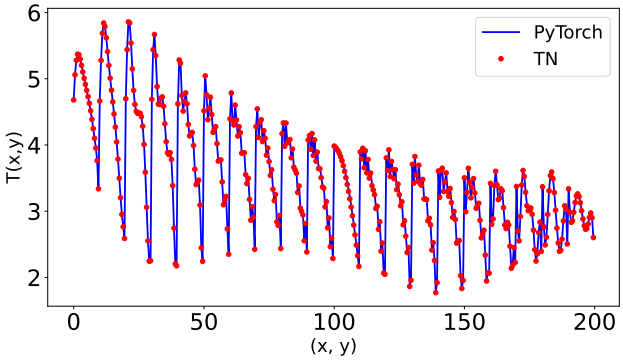

FIG. 6. Solving the static two dimensional heat equation with sources with equation (28). In blue the inversion performed with PyTorch and in red the inversion performed with the tensor network.

The root mean square error of our tensor network from the exact inversion was 10^{-4} and took 4.64 seconds to run, compared to 1.98 ms of the PyTorch method.

D. Comparison against original HHL

Now, we are going to compare the results of our TN HHL against an implementation of the original HHL al-

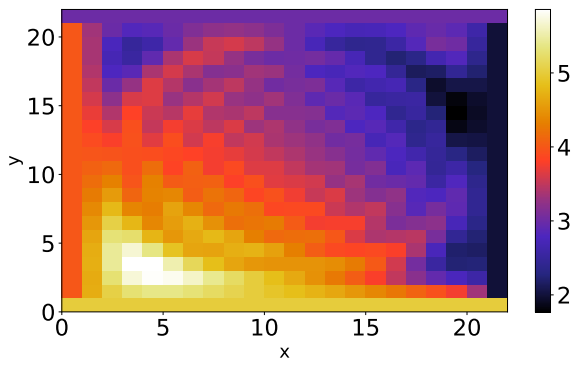

FIG. 7. Solving the static two dimensional heat equation with sources with equation (28).

gorithm in Qiskit, simulated in AerSimulator. We will generate 20 random sparse matrices of dimension 16×16, which implies 4 state qubits, and use hyperparameters $n_c = 20$, $C = 1$, $t = 1$ for original HHL and $t = 100$ for TN HHL, because they provide the best results for every method. we will check which method obtains the best result for every matrix. The correct result is also obtained by the PyTorch inversion function. The matrices have a non-diagonal density of 25%, with all their elements generated in the interval $(-1, 1)$, the same for the vectors $\vec{b}$. After their generation, the matrices are transformed to symmetric matrices and normalized by their largest eigenvalue.

In the tests, we have obtained a mean root mean square error for the TN HHL of 7.5×10^{-3}, against 2.4×10^{-2} of the original HHL. The mean root mean square error between the TN HHL and the original HHL is 1.047. In almost all of the experiments, the TN HHL obtained a better result than the original HHL. These results imply that the TN HHL avoids the errors of the original HHL to obtain a better solution. The TN HHL computes extremely faster than the original HHL, but this is due to our inefficient implementation of the original HHL.

VII. CONCLUSIONS

We have seen that our algorithm offers a way to invert matrices, solve linear equations, and perform numerical simulations based on it. We have also observed that its scaling is remarkably good with the size of the matrix to be inverted, while it can be realized on classical computers and accelerated with GPUs.

An advantage of this method is that it allows one to observe the best possible theoretical result due to a quantum HHL, as it simulates what should happen without gate errors, post-selection problems, or inaccuracies in state creation, providing a lower bound.

However, we have observed that the effective computational speed is remarkably low compared to methods already implemented in libraries such as PyTorch or Numpy, mainly due to the creation time of the tensors.

Future research in this area might aim to enhance the overall efficiency of the method by leveraging the specific properties of the used tensors, optimizing the parallelization of calculations, incorporating tensor network compression techniques, tailoring the approach to tridiagonal matrices, or adapting it for complex eigenvalues.

ACKNOWLEDGEMENTS

The research leading to this paper has received funding from the Q4Real project (Quantum Computing for Real Industries), HAZITEK 2022, no. ZE-2022/00033.

[1] A. Schwarzenberg-Czerny, On matrix factorization and efficient least squares solution., Astronomy and Astrophysics Supplement Series **110**, 405 (1995).
[2] G. Shabat, Y. Shmueli, Y. Aizenbud, and A. Averbuch, Randomized lu decomposition, Applied and Computational Harmonic Analysis **44**, 246 (2018).
[3] R. Mittal and A. Al-Kurdi, Lu-decomposition and numerical structure for solving large sparse nonsymmetric linear systems, Computers & Mathematics with Applications **43**, 131 (2002).
[4] M. R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, Journal of research of the National Bureau of Standards **49**, 409 (1952).
[5] M. Verschoor and A. C. Jalba, Analysis and performance estimation of the conjugate gradient method on multiple gpus, Parallel Computing **38**, 552 (2012).
[6] A. W. Harrow, A. Hassidim, and S. Lloyd, Quantum algorithm for linear systems of equations, Phys. Rev. Lett. **103**, 150502 (2009).
[7] A. Zaman, H. J. Morrell, and H. Y. Wong, A step-by-step hhl algorithm walkthrough to enhance understanding of critical quantum computing concepts, IEEE Access **11**, 77117 (2023).
[8] Y. Wang, Z. Hu, B. C. Sanders, and S. Kais, Qudits and high-dimensional quantum computing, Frontiers in Physics **8**, 10.3389/fphy.2020.589504 (2020).
[9] J. Biamonte and V. Bergholm, Tensor networks in a nutshell (2017), arXiv:1708.00006 [quant-ph].
[10] R. Orús, A practical introduction to tensor networks: Matrix product states and projected entangled pair states, Annals of Physics **349**, 117 (2014).
[11] H.-M. Rieser, F. Köster, and A. P. Raulf, Tensor networks for quantum machine learning, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences **479**, 10.1098/rspa.2023.0218 (2023).
[12] J. R. Pareja Monturiol, D. Pérez-García, and A. Pozas-Kerstjens, Tensorkrowch: Smooth integration of tensor networks in machine learning, Quantum **8**, 1364 (2024).

[13] H. Chen and T. Barthel, Machine learning with tree tensor networks, cp rank constraints, and tensor dropout, IEEE Transactions on Pattern Analysis and Machine Intelligence **46**, 7825 (2024).

[14] M. Lubasch, P. Moinier, and D. Jaksch, Multigrid renormalization, Journal of Computational Physics **372**, 587 (2018).

[15] N. Gourianov, M. Lubasch, S. Dolgov, Q. Y. van den Berg, H. Babaee, P. Givi, M. Kiffner, and D. Jaksch, A quantum-inspired approach to exploit turbulence structures, Nature Computational Science **2**, 30 (2022).

[16] Note sur une méthode de résolution des équations normales provenant de l'application de la méthode des moindres carrés a un système d'équations linéaires en nombre inférieur a celui des inconnues. — application de la méthode a la résolution d'un système defini d'équations linéaires, Bulletin géodésique **2**, 67 (1924).

[17] R. S. V. and, Iterative methods for solving matrix equations, The American Mathematical Monthly **72**, 67 (1965), https://doi.org/10.1080/00029890.1965.11970700.

[18] Y. Saad and M. H. Schultz, Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM Journal on Scientific and Statistical Computing **7**, 856 (1986), https://doi.org/10.1137/0907058.

[19] H. A. van der Vorst, Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems, SIAM Journal on Scientific and Statistical Computing **13**, 631 (1992), https://doi.org/10.1137/0913035.

[20] X.-M. Zhang, T. Li, and X. Yuan, Quantum state preparation with optimal circuit depth: Implementations and applications, Physical Review Letters **129**, 10.1103/physrevlett.129.230504 (2022).

[21] D. W. Berry, G. Ahokas, R. Cleve, and B. C. Sanders, Efficient quantum algorithms for simulating sparse hamiltonians, Communications in Mathematical Physics **270**, 359–371 (2006).

[22] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge Series on Information and the Natural Sciences (Cambridge University Press, 2000).

[23] D. Coppersmith, An approximate fourier transform useful in quantum factoring (2002), arXiv:quant-ph/0201067 [quant-ph].