Towards identifying possible fault-tolerant advantage of quantum linear system algorithms in terms of space, time and energy

Yue Tu,[1,2] Mark Dubynskyi,[3,4] Mohammad Mohammadisiahroudi,[5] Katia Riashchentceva,[4] Jinglei Cheng,[1] Dmitry Ryashchentse,[6] Tamás Terlaky,[5] and Junyu Liu[1]

[1]*Department of Computer Science, The University of Pittsburgh, Pittsburgh, PA 15260, USA*
[2]*Department of Computational and Applied Mathematics, The University of Chicago, Chicago, IL 60637, USA*
[3]*Mason Experimental Geometry Lab, Fairfax, VA 22030, USA*
[4]*The University of Oregon, Eugene, OR 97403, USA*
[5]*Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015, USA*
[6]*Charles Schwab Corporation, Westlake, TX 76262, USA*

(Dated: February 15, 2025)

Quantum computing, a prominent non-Von Neumann paradigm beyond Moore's law, offers superpolynomial speedups over classical methods for specific problems. However, its practical advantages in terms of space, time, and energy efficiency remain unclear. Furthermore, its applicability to critical real-world problems in science and industry, such as machine learning, is still uncertain. Finally, quantum noise in current devices entails significant overheads for error correction and fault tolerance, complicating fair comparisons with their classical counterparts. In this paper, we address all three issues through a detailed resource estimation of space, time, and energy for fault-tolerant superconducting quantum devices. We focus on the Harrow–Hassidim–Lloyd (HHL) algorithm, a quantum linear system algorithm with practical applications in machine learning. For the pure computing process of these algorithms (excluding memory and data transfer) and compared to the exact classical counterpart (the conjugate gradient method), we find that possible quantum advantages could emerge around the matrix size $N \approx 2^{33} \sim 2^{48}$, requiring $\mathcal{O}(10^5)$ physical qubits, $\mathcal{O}(10^{12} \sim 10^{13})$ Joules of energy, and approximately 10 days of computational time ($\mathcal{O}(10^6)$ seconds) under precise settings of surface code fault-tolerant computation and magic state distillation, with condition number, sparsity, and precision $\kappa, s \approx \mathcal{O}(10 \sim 100), \epsilon \sim 0.01$. Key parameters for the algorithm, including N, κ, s, ϵ, are all adjustable in our resource estimator. A precise map of the quantum-classical comparison for these parameters is provided, illustrating the boundary of a practical, potential quantum advantage. Our work demonstrates that significant benefits from quantum computing in important real-world problems, such as quantum linear system algorithms, are achievable and quantitatively determines how advanced a fault-tolerant quantum computer must be to realize this goal.

I. INTRODUCTION

Quantum computing [1] stands as the leading paradigm of next-generation computing technology. Unlike traditional computing, quantum computers can access and manipulate quantum states, which serve as carriers of information for computational purposes. One of the primary motivations for developing quantum computing is its potential to achieve a *possible quantum advantage* over its classical counterparts in solving certain problems. This potential quantum advantage, studied primarily in terms of time cost within computational complexity theory, has been theoretically demonstrated in problems such as factoring [2], searching [3], and simulation [4]. These advancements offer hope for sustaining or surpassing Moore's law in the semiconductor industry.

However, beyond the time complexity estimated in the gate models from theoretical computer science, it is challenging to estimate and justify the possible quantum advantage in practice. First, practical cost estimation of quantum computing requires state-of-the-art knowledge, from detailed theory covering prefactors in front of the big-O notation of complexity [5, 6], to explicit designs of quantum hardware, and it includes more comprehensive measurement such as time costs (measured in seconds), space costs (number of physical qubits), and in particular energy costs. Especially, the complex nature of quantitative energy sustainability estimation is highly uninvestigated, although the possible energy advantage of quantum computing algorithm is discussed mostly in the qualitative argument [7, 8]. Second, although the existence of potential quantum advantage for some algorithms is solidly justified in theory, it is challenging to prove that those algorithms can turn into real-world, significant benefits especially for commercial applications [9]. Finally, quantum states are extremely fragile and current quantum processors are noisy, making quantum error correction the only possible way to make large-scale, fault-tolerant quantum computing. Fault-tolerance, although sustainable in theory, requires lots of additional resources and experimental challenges, making precise resource estimation much more challenging.

In this work, we address all three of these challenges

by conducting a full-stack, energy-aware resource estimation for the so-called Harrow–Hassidim–Lloyd (HHL) algorithm [10]. The HHL algorithm provides a Quantum Linear System Algorithm (QLSA) that can be used for matrix inversion. Given a linear equation $A|x\rangle = |b\rangle$, the algorithm returns a quantum state $|x\rangle = A^{-1}|b\rangle$ as a solution. For certain classes of matrices, it has been theoretically shown that the algorithm runs in $\log N$ time for an $N \times N$ matrix, making it exponentially faster than any known classical counterpart. Since linear algebra applications are ubiquitous in modern science and technology, the algorithm is regarded as one of the most promising applications for large-scale, fault-tolerant quantum processors [11].

To identify the regime of possible, practical quantum advantages, we systematically investigate the costs of QLSA. The primary input of QLSA is a general Hermitian matrix A of dimension $N = 2^n$, with row sparsity s (where $s = \max_j(\ \#\text{ of nonzero entries in row } j \text{ of } A)$), precision ϵ, and condition number κ. For non-Hermitian matrices, one can effectively solve a corresponding Hermitian matrix instead, requiring only one additional qubit [10, 11]. At the logical level, we primarily adopt the original and classical HHL framework [10, 11]. For quantum simulation, we employ the one-sparse Hamiltonian simulation model to implement e^{iAt}, using [12] to estimate the Trotter error. Finally, we provide an explicit formula for estimating resource requirements at the logical level for HHL, particularly focusing on the number of different types of quantum gates. These estimates are validated using a quantum circuit state-vector simulation code.

Moreover, we extend our analysis to resource estimations at the fault-tolerant level. To achieve this, we utilize the framework proposed by Litinski [13] to estimate the fault-tolerance resource. The framework assume clifford gates are cheap and only accepts the number of T-gates as input, then search for the optimal surface code scheme that will minimize the space-time cost with logical error rate below one percent. . Finally, we compare our fault-tolerant resource estimates with those of classical algorithms. In particular, we discuss the classical counterpart to the Quantum Linear System Algorithm (QLSA): the conjugate gradient (CG) method [14]. The CG method is widely regarded as the classical analogue of QLSA, as it similarly exploits the sparsity and well-conditioned nature of the problem. Its time complexity is given by $O(Ns\kappa \log(1/\epsilon))$. We conduct a detailed analysis of its exact scaling behavior and provide an estimation of its clock cycle requirements.

The primary energy costs associated with executing the HHL algorithm stem from the cooling system. On the quantum side, we estimate the energy consumption based on the model proposed in [15], assuming that the energy cost scales linearly with the number of qubits, the total energy cost is given by the product of the runtime, the number of physical qubits, and a cooling efficiency factor.

To estimate the cooling efficiency, we refer to [16], which provides an approximation based on IBM Quantum System Two's dilution refrigerator, with an estimated energy efficiency of 6.25 Watts per qubit. However, it is important to note that cooling efficiency is highly dependent on the quantum computer's architecture and may vary significantly as quantum hardware evolves. Consequently, this estimation should be viewed as a rough approximation rather than a precise metric.

On the classical side, we estimate the energy consumption using the power requirements of a typical desktop CPU. A standard desktop processor operating at a clock frequency of 1 GHz consumes approximately 50 Watts. This provides a baseline for evaluating the comparative energy efficiency of classical and quantum computations in the context of the QLSA algorithm.

Note that we consider only the costs incurred during the computation process. The costs associated with the interface between classical and quantum processors—such as uploading the matrix A and the vector $|b\rangle$ to the quantum device, as well as downloading the state $|x\rangle$ to classical memory—are not addressed in this work. The uploading problem can be mitigated by fast quantum memory solutions, such as Quantum Random Access Memory (QRAM) [17], whose circuits are, in fact, classically simulatable [18]. Meanwhile, the downloading process depends on the specific needs of the user. Since classical users cannot handle exponentially large datasets [19], quantum tomography techniques [20, 21] can be employed to extract partial information from $|x\rangle$.

Our work is organized as follows. In Section II, we discuss our results comparing quantum and classical approaches, along with their implications for computational speed and energy efficiency. In Section III, we outline the key methodologies employed to obtain these results. At the logical level, this includes circuit implementation and exact scaling derivation, while at the physical level, it encompasses the implementation of the surface code optimizer. In Section IV, we provide conclusions and an outlook on future research directions. Additional technical details, including a detailed formulation of the HHL algorithm, Trotter simulation strategies, surface code setups, classical counterparts, and energy analysis, are provided in the Supplementary Material.

II. RESULTS

In this chapter, we present the main conclusions of our study, structured in two parts. The first part examines the operation count overhead associated with the algorithm's execution, independent of hardware parameters. For the quantum component, this is quantified by the logical T-gate count—the bottleneck logical resource in surface code-based quantum computing—while the classical component is evaluated based on the number of floating-point operations (FLOPs).

Proposition 1. *Consider a system of linear equations $Ax = b$, where A has been scaled such that its eigenvalues lie in the interval $[\frac{1}{\kappa}, 1]$. Denote by $|b\rangle$ the normalized quantum state proportional to b. Assume access to an oracle that provides access to the elements of a sparse submatrix of A. Then, there exists a quantum algorithm that takes the input state $|b\rangle$ and outputs the normalized solution state $|\tilde{x}\rangle$ with additive error $||\tilde{x} - x||$ less than ϵ. The algorithm requires T T-gates and Q queries to the oracle, where:*

$$T = \frac{\sqrt{\frac{320}{3}}\pi\kappa^2 s}{\epsilon^2} \times (18\log N + 90r + 15), \tag{1}$$

$$Q = \frac{\sqrt{\frac{320}{3}}\pi\kappa^2 s}{\epsilon^2} \times 2, \tag{2}$$

where N is the matrix size, κ is the condition number, s is the sparsity of A, and ϵ is the error.

We refer the reader to appendix A for a detailed derivation of 1. Here, we emphasize that 1 exhibit different scaling compared to the results presented in the original paper, where the complexity was given by $O\left(\log(N) \cdot s^2\kappa^2/\epsilon\right)$. In our work, we have increased the dependence on ϵ to order 2 while reducing the dependence on s to order 1.

The dependence on ϵ is because we comprehensively analised the error introduced by trotterization and QFT, while the original paper's error estimation only account for the latter. For detailed derivation we refer readers to B.

The reduction in s-dependence is achieved by assuming that the matrix is perfectly decomposed into exactly s sub matrices at the oracle level. Although this assumption does not hold for general sparse matrices, many QLSA applications involve fixed matrix structures, which allow for efficient oracle implementation that satisfies this assumption, for example, in [22] and [23]. If we remove this assumption, we'll have to additionally implement the decomposition procedure, one approach is to utilize graph coloring algorithm, which decompose the matrix into $6s^2$ one sparse matrices, thus increase a prefactor of 6 and square the dependence on sparsity. Furthermore, the graph coloring procedure is being called in each HS1, and according to our implementation, the graph coloring procedure involve around $200 \times \log N$ T-gates, this cost is about 10 times the other cost in HS1 ($18\log N + 90r + 15$), thus introducing another prefactor of 10, so overall, the cost for graph coloring based general QLSA will have a scaling of $60sT$.

Proposition 2. *For the CG method solving the linear system $Ax = b$, the algorithm runs in C FLOPs, where:*

$$C = (4Ns + 14N) \times \left(\frac{1}{2}\kappa \log\left(\frac{2}{\epsilon}\right)\right) \tag{3}$$

N is the matrix size, k is the condition number, s is the sparsity, and ϵ is the error threshold.

The detailed derivation of 2 we refer readers to appendix 4

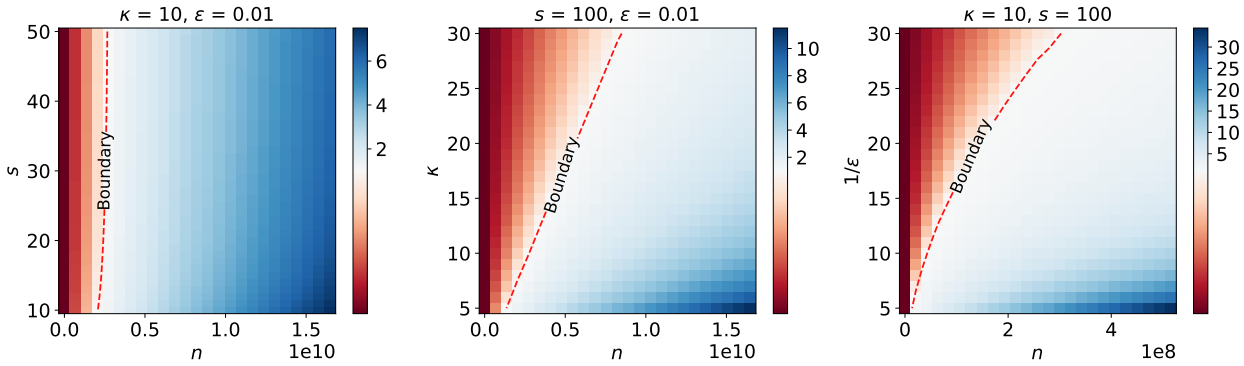
Next, we incorporate hardware parameters to illustrate the specific runtime overhead and its relationship with various input parameters. The assumed hardware parameters are as follows: the quantum computer has an infinite number of physical qubits, a physical qubit error rate of 10^{-5}, a logical cycle execution time of 10 ns, and an energy consumption of 6.25 watts per physical qubit. We present heatmaps 1 and 1, which illustrate the runtime and energy cost ratio between classical and quantum algorithms. In these heatmaps, blue regions indicate a ratio less than 1, implying that the quantum algorithm incurs lower overhead than its classical counterpart. Consequently, these regions can be interpreted as exhibiting possible quantum advantage. Also, we provide line charts 1, 1, and 1 to depict how runtime, space, and energy costs vary with matrix size, where we assume conditional number to be fixed to 10, the sparsity to be $\log N$. Of course, this assumption may not perfectly align with future quantum computers. For readers interested in different configurations, the runtime estimation and energy consumption can be easily rescaled based on the ratio of the new parameters for clock cycle time and energy consumption per qubit.

III. METHODS

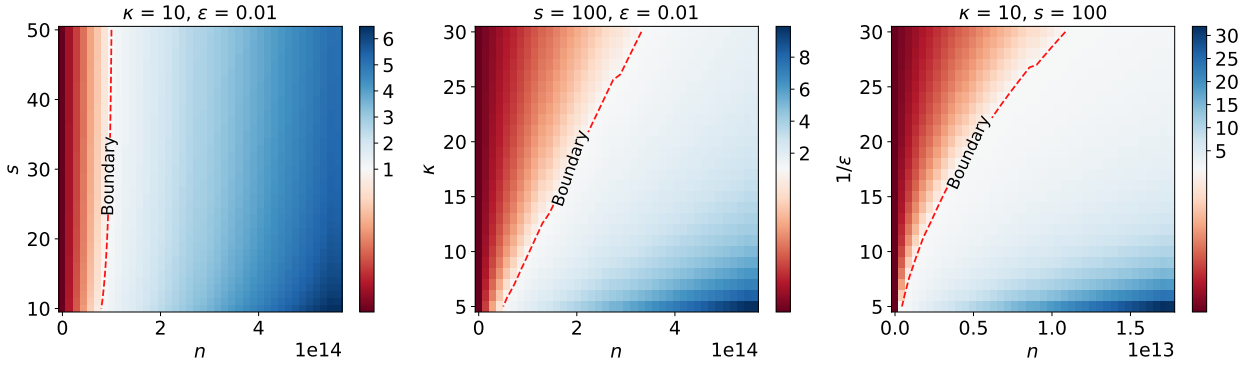
A. Quantum part

In this work, we present the first end-to-end analysis of the QLSA, covering the resource cost from logical level to runtime-level. The overall workflow of this study is illustrated in Fig. 1 and is broadly divided into two stages: logical resource analysis and physical resource analysis.

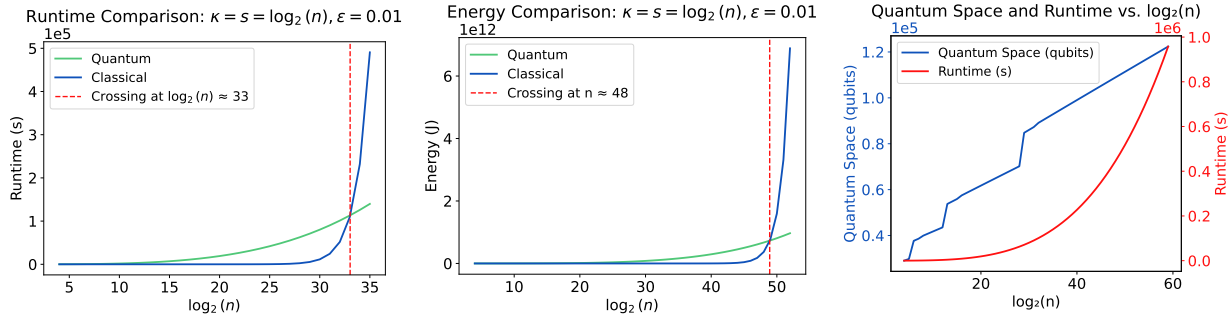
At the logical level, we first implemented the quantum circuit for the Quantum Linear Systems Algorithm (QLSA) using Q#, a programming language designed for circuit-based quantum computing. This implementation allowed us to estimate the logical qubit and T-gate requirements for each submodule. Among all submodules in the QLSA algorithm, the primary bottleneck lies in one specific subroutine—the one-sparse Hamiltonian simulation (HS1). This subroutine is responsible for constructing a unitary operation that implements e^{iAt}, where A is the input matrix. The high resource demand of HS1 stems from the Trotterization process, which requires applying HS1 repeatedly to suppress the error to a desired level. Additionally, the unitary operation itself must be executed multiple times as part of the Quantum Phase Estimation (QPE) procedure, further increasing the total number of HS1 invocations. To precisely determine the scaling behavior of HS1, it is essential to analyze the algorithm's error. There are two primary sources of error in the QLSA algorithm: (1) errors introduced by Trotterization and (2) errors from the Quantum Fourier Transform (QFT). While previous analyses of QLSA have



(a) Runtime Ratio: Classical / Quantum, the blue region corresponds to possible quantum advantage.



(b) Energy Ratio, Classical / Quantum, the blue region corresponds to possible quantum advantage.



(c) Runtime Comparison (d) Energy Comparison (e) Runtime and Space Cost

FIG. 1: **Comparison of Quantum Linear System Algorithm (QLSA) and Classical Conjugate Gradient Algorithm.** In the plots, N represents matrix size, s is the sparsity, k is the condition number, ϵ is the error tolerance in terms of vector-2 norm. (a) & (b): Heatmaps showing the ratio of classical runtime to quantum runtime and classical energy consumption to quantum energy consumption under different problem parameters. Blue regions indicate a possible quantum advantage. (c) & (d): Runtime and energy comparison between quantum and classical algorithms, assuming condition number and sparsity scale as $\log_2(N)$, and ϵ to be 0.01. (e): Runtime and qubit count scaling for the quantum algorithm over larger matrix sizes.

not explicitly addressed Trotterization, our work provides the first comprehensive error analysis that accounts for both error sources simultaneously. Through mathematical derivation, we obtained an exact upper bound for the total error and found that this bound is significantly higher than the average-case error. To refine this analysis, we conducted numerical experiments to give a probabilistic bound, revealing that the expected cost of the HHL algorithm is substantially lower than its worst-case estimate.

Based on our error estimation, we determined the required number of HS1 repetitions for specific input cases. The gate count per HS1 operation was obtained from our earlier code implementation. Since T-gates represent the primary resource bottleneck in our analysis, we focus on tracking only the T-gate count. The total number of T-gates required is given by the product of the total number of HS1 invocations and the T-gate count per HS1 operation, which give us result as shown in 1.

After determining the number of logical qubits and logical T-gates required by the algorithm, the next step is to translate these logical resources into the fault-tolerant physical resource under a quantum error correction (QEC) scheme. Among the various existing QEC schemes, we adopt the surface code in this study as it has the highest fault tolerant error threshold, and has been demonstrated by recent experiments [24].

In essence, when executing quantum algorithms using the surface code model, physical qubits are categorized into data blocks and distillation blocks. Data blocks are responsible for storing logical qubits and consuming the so-called "magic states," while distillation blocks generate these magic states. Each time a data block consumes a magic state produced by a distillation block, a T-gate operation is executed. Consequently, the total physical qubit count is simply the sum of physical qubits in the data and distillation blocks. The runtime of surface code-based quantum computing is approximately given by the number of T-gates divided by their execution speed, which is determined by the specific configuration of data and distillation blocks as well as the hardware parameters of the quantum computer.

To determine the physical qubit count and runtime, it is essential to establish an optimal configuration of data and distillation blocks. Our goal is to find a configuration that minimizes the space-time volume, defined as the product of the number of physical qubits and the runtime. Achieving this requires considering several constraints and trade-offs, which are listed as follows:

- The distillation block must be large enough to achieve the desired T-state error rate.
- The code distance of the surface code must be large enough to achieve the desired logical error rate.
- The data block's efficiency to consume magic states comes at a price of using more physicla qubits.
- The distillation block's efficiency to produce magic states comes at a price of using more physical qubits.

We use the procedure proposed in [13] to optimize space-time volume under such considerations. The procedure includes 4 steps:

1. Determine the distillation protocol. We select the most cost-effective distillation protocol that achieves the required T-state error rate.
2. Construct a minimal setup. We design the space-optimal protocol.
3. Determine the code distance. We identify the minimum code distance that ensures the error rate remains below the threshold.
4. Add distillation blocks. Building on the space-optimal protocol, we incrementally adopt data block and distillation block protocols with larger space overhead but reduced time overhead until the optimal space-time cost is achieved.

We implemented these four steps in a script, which will be further discussed in Appendix. By inputting the algorithm's logical T-gate count, logical qubits count, and reasonably assumed quantum computer parameters (including quantum gate fidelity, the number of physical qubits, and clock frequency), the script identifies the configuration of distillation and data blocks that minimizes space-time volume. It then outputs the corresponding runtime and physical qubits overhead.

So now we are only left with energy estimation. We use the model proposed in [15], where the primary energy cost of quantum computers is cooling, and the energy consumption grows with number of qubits because they assume that a single refrigeration unit can accommodate a limited number of qubits for super-conducting qubits. As a result, the quantum computer's power consumption is proportional to the number of physical qubits. Now we define the energy efficiency as the power consumption per qubit, we use the data provided by [16], where they targeted at IBM Quantum System Two's dilution refrigerator, the system houses 4158 physical qubits and operates at a power of 27 KW, thus, in interms of single qubit, the power efficiency is 6.25 Watts.

These results provide an intuitive and detailed reference for end-to-end runtime estimation of the resources required to run the QLSA algorithm and serve as a practical basis for developing QLSA-related applications. However, we emphasize that current quantum computers are far from being capable of running the HHL algorithm, and future quantum hardware may differ significantly from existing systems. Also, there has been significant progress in the QLSA algorithm, such that the algorithm we are esitmating may not be the optimal. Therefore, the specific results we provide should be viewed as rough baseline predictions based on the current state of quantum computing technology.

B. Classical part

In this work, we address two classical linear system solver: the conjugate gradient (CG) method and the Cholesky decomposition (CD) method. We mainly focus on former as CG is generally considere as the classical counterparat of QLSA due to its iterative nature and ability to handle sparse and well-conditioned systems. on

the other hand, the CD method is not a direct correspondence to QLSA as it is deterministic with no dependence on ϵ, however CD is more often used in supercomputers and we include it in D.

For detailed derivation of the resource cost for boths methods we refer readers to appendix D.

IV. CONCLUSION AND OUTLOOK

In this work, we perform an end-to-end resource estimation of the HHL algorithm in terms of time, space, and energy. Unlike existing resource estimations, including [5], our work provides a detailed analysis of fault tolerance based on surface codes. Moreover, we precisely address energy sustainability by employing physical models of superconducting quantum devices. Our findings confirm that quantum computing will indeed have an energy advantage at a large scale based on current superconducting quantum technologies, resolving a long-standing concern about the energy sustainability of quantum computing in practical applications [7, 8]. Other innovative aspects of our work include a detailed comparison between quantum and classical approaches to solving linear systems and matrix inversion, an analysis of theoretical upper bounds versus practical average cases, and a quantum circuit simulator for end-to-end HHL algorithms, the details of which are summarized in the Supplementary Materials.

Although we confirm that potential quantum advantages might appear at the scale of $2^{35} \sim 2^{50}$ matrix sizes and 10^5 physical qubits, the practical costs could be lower due to the overestimation of resource counts in the Trotter simulation. Moreover, smarter designs for quantum error correction and energy-sustainable innovations in quantum hardware might further enhance performance. On the other hand, compared to other resource estimations of quantum algorithms in different domains (such as cryptography used for digital signatures and blockchains [25]), the HHL algorithm appears to achieve potential quantum advantages with a smaller number of physical qubits [25]. However, better designs, including quantum memories, might further improve the capabilities of cryptographically relevant quantum computers [26].

Our work presents significant opportunities for bringing quantum computing into practice. For example, we do not address the uploading and downloading challenges of the HHL algorithm, which may be critical for data-intensive applications. Active developments are underway on both the hardware side [27] and the system design side [28, 29] toward large-scale and error-resilient QRAMs [18] in promising hardware platforms such as circuit/cavity QED/QAD [27]. Notably, since QRAM circuits are classically simulatable, precise estimates of QRAM costs can be investigated on a large scale [18]. Moreover, for the downloading process, a detailed classification of quantum computing user needs may be essential. If quantum customers are classical, what will classical customers require? The first few elements of the state $|x\rangle$? Some expectation value of operators [20]? Other important directions include further exploration of energy sustainability. Can we provide energy estimations for other hardware, such as neutral-atom quantum computers? Can we make device-independent statements about energy advantages? Can we generalize such estimations to a broader range of quantum algorithms? Finally, it would also be interesting to investigate more advanced versions of quantum algorithms [30–34] and classical algorithms [14], as well as improved designs for quantum error correction codes [35–37] and quantum memories [26].

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

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

[3] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[4] Seth Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–1078, 1996.

[5] David Jennings, Matteo Lostaglio, Sam Pallister, Andrew T Sornborger, and Yiğit Subaşı. Efficient quantum linear solver algorithm with detailed running costs. *arXiv preprint arXiv:2305.11352*, 2023.

[6] Alexander M Dalzell, B David Clader, Grant Salton, Mario Berta, Cedric Yen-Yu Lin, David A Bader, Nikitas Stamatopoulos, Martin JA Schuetz, Fernando GSL Brandão, Helmut G Katzgraber, et al. End-to-end resource analysis for quantum interior-point methods and portfolio optimization. *PRX Quantum*, 4(4):040325, 2023.

[7] Alexia Auffèves. Quantum technologies need a quantum energy initiative. *PRX Quantum*, 3(2):020101, 2022.

[8] Travis L Scholten, Carl J Williams, Dustin Moody, Michele Mosca, William Hurley, William J Zeng, Matthias Troyer, Jay M Gambetta, et al. Assessing the benefits and risks of quantum computers. *arXiv preprint arXiv:2401.16317*, 2024.

[9] John Preskill. *Beyond NISQ: The Megaquop Machine*, Q2B 2024.

[10] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.

[11] Danial Dervovic, Mark Herbster, Peter Mountney, Simone Severini, Naïri Usher, and Leonard Wossnig. Quantum linear systems algorithms: a primer. *arXiv preprint arXiv:1802.08227*, 2018.

[12] Andrew M Childs, Yuan Su, Minh C Tran, Nathan

Wiebe, and Shuchen Zhu. Theory of trotter error with commutator scaling. *Physical Review X*, 11(1):011020, 2021.

[13] Daniel Litinski. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum*, 3:128, 2019.

[14] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

[15] Michael James Martin, Caroline Hughes, Gilberto Moreno, Eric B. Jones, David Sickinger, Sreekant Narumanchi, and Ray Grout. Energy use in quantum data centers: Scaling the impact of computer architecture, qubit performance, size, and thermal parameters, 2021.

[16] Edward Parker and Michael JD Vermeer. Estimating the energy requirements to operate a cryptanalytically relevant quantum computer. *arXiv preprint arXiv:2304.14344*, 2023.

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

[18] Connor T Hann, Gideon Lee, SM Girvin, and Liang Jiang. Resilience of quantum random access memory to generic noise. *Prx Quantum*, 2(2):020311, 2021.

[19] Scott Aaronson. Read the fine print. *Nature Physics*, 11(4):291–293, 2015.

[20] Hsin-Yuan Huang, Richard Kueng, and John Preskill. Predicting many properties of a quantum system from very few measurements. *Nature Physics*, 16(10):1050–1057, 2020.

[21] Junyu Liu, Minzhao Liu, Jin-Peng Liu, Ziyu Ye, Yunfei Wang, Yuri Alexeev, Jens Eisert, and Liang Jiang. Towards provably efficient quantum algorithms for large-scale machine-learning models. *Nature Communications*, 15(1):434, 2024.

[22] Jin-Peng Liu, Herman Øie Kolden, Hari K. Krovi, Nuno F. Loureiro, Konstantina Trivisa, and Andrew M. Childs. Efficient quantum algorithm for dissipative nonlinear differential equations. *Proceedings of the National Academy of Sciences*, 118(35), August 2021.

[23] Junyu Liu, Minzhao Liu, Jin-Peng Liu, Ziyu Ye, Yunfei Wang, Yuri Alexeev, Jens Eisert, and Liang Jiang. Towards provably efficient quantum algorithms for large-scale machine-learning models. *Nature Communications*, 15(1), January 2024.

[24] Rajeev Acharya, Laleh Aghababaie-Beni, Igor Aleiner, Trond I. Andersen, Markus Ansmann, Frank Arute, Kunal Arya, Abraham Asfaw, Nikita Astrakhantsev, Juan Atalaya, Ryan Babbush, Dave Bacon, Brian Ballard, Joseph C. Bardin, Johannes Bausch, Andreas Bengtsson, Alexander Bilmes, Sam Blackwell, Sergio Boixo, Gina Bortoli, Alexandre Bourassa, Jenna Bovaird, Leon Brill, Michael Broughton, David A. Browne, Brett Buchea, Bob B. Buckley, David A. Buell, Tim Burger, Brian Burkett, Nicholas Bushnell, Anthony Cabrera, Juan Campero, Hung-Shen Chang, Yu Chen, Zijun Chen, Ben Chiaro, Desmond Chik, Charina Chou, Jahan Claes, Agnetta Y. Cleland, Josh Cogan, Roberto Collins, Paul Conner, William Courtney, Alexander L. Crook, Ben Curtin, Sayan Das, Alex Davies, Laura De Lorenzo, Dripto M. Debroy, Sean Demura, Michel Devoret, Agustin Di Paolo, Paul Donohoe, Ilya Drozdov, Andrew Dunsworth, Clint Earle, Thomas Edlich, Alec Eickbusch, Aviv Moshe Elbag, Mahmoud Elzouka, Catherine Erickson, Lara Faoro, Edward Farhi, Vinicius S. Ferreira, Leslie Flores Burgos, Ebrahim Forati, Austin G. Fowler, Brooks Foxen, Suhas Ganjam, Gonzalo Garcia, Robert Gasca, Élie Genois, William Giang, Craig Gidney, Dar Gilboa, Raja Gosula, Alejandro Grajales Dau, Dietrich Graumann, Alex Greene, Jonathan A. Gross, Steve Habegger, John Hall, Michael C. Hamilton, Monica Hansen, Matthew P. Harrigan, Sean D. Harrington, Francisco J. H. Heras, Stephen Heslin, Paula Heu, Oscar Higgott, Gordon Hill, Jeremy Hilton, George Holland, Sabrina Hong, Hsin-Yuan Huang, Ashley Huff, William J. Huggins, Lev B. Ioffe, Sergei V. Isakov, Justin Iveland, Evan Jeffrey, Zhang Jiang, Cody Jones, Stephen Jordan, Chaitali Joshi, Pavol Juhas, Dvir Kafri, Hui Kang, Amir H. Karamlou, Kostyantyn Kechedzhi, Julian Kelly, Trupti Khaire, Tanuj Khattar, Mostafa Khezri, Seon Kim, Paul V. Klimov, Andrey R. Klots, Bryce Kobrin, Pushmeet Kohli, Alexander N. Korotkov, Fedor Kostritsa, Robin Kothari, Borislav Kozlovskii, John Mark Kreikebaum, Vladislav D. Kurilovich, Nathan Lacroix, David Landhuis, Tiano Lange-Dei, Brandon W. Langley, Pavel Laptev, Kim-Ming Lau, Loïck Le Guevel, Justin Ledford, Kenny Lee, Yuri D. Lensky, Shannon Leon, Brian J. Lester, Wing Yan Li, Yin Li, Alexander T. Lill, Wayne Liu, William P. Livingston, Aditya Locharla, Erik Lucero, Daniel Lundahl, Aaron Lunt, Sid Madhuk, Fionn D. Malone, Ashley Maloney, Salvatore Mandrá, Leigh S. Martin, Steven Martin, Orion Martin, Cameron Maxfield, Jarrod R. McClean, Matt McEwen, Seneca Meeks, Anthony Megrant, Xiao Mi, Kevin C. Miao, Amanda Mieszala, Reza Molavi, Sebastian Molina, Shirin Montazeri, Alexis Morvan, Ramis Movassagh, Wojciech Mruczkiewicz, Ofer Naaman, Matthew Neeley, Charles Neill, Ani Nersisyan, Hartmut Neven, Michael Newman, Jiun How Ng, Anthony Nguyen, Murray Nguyen, Chia-Hung Ni, Thomas E. O'Brien, William D. Oliver, Alex Opremcak, Kristoffer Ottosson, Andre Petukhov, Alex Pizzuto, John Platt, Rebecca Potter, Orion Pritchard, Leonid P. Pryadko, Chris Quintana, Ganesh Ramachandran, Matthew J. Reagor, David M. Rhodes, Gabrielle Roberts, Eliott Rosenberg, Emma Rosenfeld, Pedram Roushan, Nicholas C. Rubin, Negar Saei, Daniel Sank, Kannan Sankaragomathi, Kevin J. Satzinger, Henry F. Schurkus, Christopher Schuster, Andrew W. Senior, Michael J. Shearn, Aaron Shorter, Noah Shutty, Vladimir Shvarts, Shraddha Singh, Volodymyr Sivak, Jindra Skruzny, Spencer Small, Vadim Smelyanskiy, W. Clarke Smith, Rolando D. Somma, Sofia Springer, George Sterling, Doug Strain, Jordan Suchard, Aaron Szasz, Alex Sztein, Douglas Thor, Alfredo Torres, M. Mert Torunbalci, Abeer Vaishnav, Justin Vargas, Sergey Vdovichev, Guifre Vidal, Benjamin Villalonga, Catherine Vollgraff Heidweiller, Steven Waltman, Shannon X. Wang, Brayden Ware, Kate Weber, Theodore White, Kristi Wong, Bryan W. K. Woo, Cheng Xing, Z. Jamie Yao, Ping Yeh, Bicheng Ying, Juhwan Yoo, Noureldin Yosri, Grayson Young, Adam Zalcman, Yaxing Zhang, Ningfeng Zhu, and Nicholas Zobrist. Quantum error correction below the surface code threshold, 2024.

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

[26] Élie Gouzien and Nicolas Sangouard. Factoring 2048-bit rsa integers in 177 days with 13 436 qubits and a multimode memory. *Physical review letters*, 127(14):140503, 2021.

[27] Connor T Hann, Chang-Ling Zou, Yaxing Zhang, Yiwen Chu, Robert J Schoelkopf, Steven M Girvin, and Liang Jiang. Hardware-efficient quantum random access memory with hybrid quantum acoustic systems. *Physical review letters*, 123(25):250501, 2019.

[28] DK Weiss, Shifan Xu, Shruti Puri, Yongshan Ding, and Steven M Girvin. Faulty towers: recovering a functioning quantum random access memory in the presence of defective routers. *arXiv preprint arXiv:2411.15612*, 2024.

[29] Yunfei Wang, Yuri Alexeev, Liang Jiang, Frederic T Chong, and Junyu Liu. Fundamental causal bounds of quantum random access memories. *npj Quantum Information*, 10(1):71, 2024.

[30] Andris Ambainis. Variable time amplitude amplification and a faster quantum algorithm for solving systems of linear equations, 2010.

[31] Andrew M. Childs, Robin Kothari, and Rolando D. Somma. Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. *SIAM J. Comp.*, 46(6):1920–1950, January 2017.

[32] Guang Hao Low and Yuan Su. Quantum linear system algorithm with optimal queries to initial state preparation, 2024.

[33] Pedro C. S. Costa, Dong An, Yuval R. Sanders, Yuan Su, Ryan Babbush, and Dominic W. Berry. Optimal scaling quantum linear systems solver via discrete adiabatic theorem, 2021.

[34] Alexander M. Dalzell. A shortcut to an optimal quantum linear system solver, 2024.

[35] Sergey Bravyi, Andrew W Cross, Jay M Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627(8005):778–782, 2024.

[36] Qian Xu, J Pablo Bonilla Ataides, Christopher A Pattison, Nithin Raveendran, Dolev Bluvstein, Jonathan Wurtz, Bane Vasić, Mikhail D Lukin, Liang Jiang, and Hengyun Zhou. Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays. *Nature Physics*, pages 1–7, 2024.

[37] Joshua Viszlai, Willers Yang, Sophia Fuhui Lin, Junyu Liu, Natalia Nottingham, Jonathan M Baker, and Frederic T Chong. Matching generalized-bicycle codes to neutral atoms for low-overhead fault-tolerance. *arXiv preprint arXiv:2311.16980*, 2023.

[38] Dominic W. Berry, Graeme Ahokas, Richard Cleve, and Barry C. Sanders. Efficient quantum algorithms for simulating sparse hamiltonians. *Communications in Mathematical Physics*, 270(2):359–371, December 2006.

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

[40] Barbara M. Terhal. Quantum error correction for quantum memories. *Rev. Mod. Phys.*, 87:307–346, Apr 2015.

[41] Earl T. Campbell, Barbara M. Terhal, and Christophe Vuillot. Roads towards fault-tolerant universal quantum computation. *Nature*, 549(7671):172–179, September 2017.

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

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

[44] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[45] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[46] TOP500.org. Top500 list - november 2024, November 2024.

[47] Wikipedia contributors. Processor power dissipation, 2024. [Online; accessed 4-February-2025].

Supplemental Material

Appendix A: QLSA algorithm implementation

1. A brief introduction to QLSA

The Quantum Linear Systems Algorithm (QLSA) is a quantum algorithm designed to solve systems of linear equations efficiently. It relies on principles of quantum mechanics to provide a possible superpolynomial advantage for specific cases compared to classical methods.

The core idea is to represent the solution to a linear system $A\mathbf{x} = \mathbf{b}$ in the quantum state $|\mathbf{x}\rangle$, where A is a sparse and well-conditioned matrix. The algorithm involves several key steps:

- **Preparation of the Input State:** The algorithm starts with the quantum encoding of the input vector $|\mathbf{b}\rangle$ into a quantum state.
- **Quantum Phase Estimation (QPE):** QPE approximate the eigenvalues of the matrix A by estimating the phase of the unitary e^{iAt}. Such unitary is implemented using the Trotter decomposition method.
- **Matrix Inversion:** The eigenvalues of A are inverted conditionally, enabling the construction of the quantum state corresponding to the solution.
- **Measurement:** Finally, measurements are performed to extract information about the solution vector.

QLSA's efficiency is highly dependent on the sparsity and condition number of the matrix A, making it particularly useful for problems where these constraints are satisfied

2. Resource bottleneck in QLSA

A significant computational bottleneck in the QLSA arises from the repetitive application of one-sparse Hamiltonian simulation, which is a fundamental component of the algorithm. This challenge emerges due to the following reasons:

- **Quantum Phase Estimation (QPE):** The QPE step requires the Hamiltonian simulation of A, which is an s-sparse matrix, to be applied multiple times to achieve sufficient precision. The repetition ensures the accurate estimation of eigenvalues, which directly affects the correctness of the algorithm.
- **Simulation of s-Sparse Hamiltonians:** To implement the simulation of s-sparse Hamiltonians, the Trotter decomposition method is often employed. However, this approach introduces further overhead, as it decomposes the s-sparse Hamiltonian into a series of one-sparse Hamiltonians.
- **One-Sparse Hamiltonian Simulation:** Each s-sparse Hamiltonian simulation requires the repetitive application of one-sparse Hamiltonian simulations. This nested structure compounds the computational cost, making one-sparse Hamiltonian simulation a critical bottleneck for resource efficiency.

In essence, the hierarchical nature of the QLSA—where QPE requires multiple iterations of s-sparse simulations, and s-sparse Hamiltonian simulations depend on repeatedly applying a series of one-sparse simulations—highlights the algorithm's computational resource bottleneck.

Therefore, our resource estimation focuses solely on the one-sparse Hamiltonian simulations, reducing the overall resource estimation to (1) the number of one-sparse Hamiltonian simulations. (2) the resource cost of each one-sparse Hamiltonian simulation. Next we will discuss the resource cost of each one-sparse Hamiltonian simulation by giving a concrete code implementation of such step. We left the analysis for number of one-sparse Hamiltonian simulation in the error analysis section.

3. One-sparse Hamiltonian simulation implementation

In general, directly implementing the unitary e^{iHt} is infeasible because H is an extremely large matrix, and computing its exponential is highly complex. Here we use the methods in [38]. The core idea for implementing e^{iHt} relies

on the *unitary conjugation theorem*, which states that for a matrix exponential of the form $e^{iUHU^\dagger}$, it is equivalent to $Ue^{iH}U^\dagger$.

Thus, if we can decompose H into the form $U_1U_2\dots H'\dots U_2^\dagger U_1^\dagger$, where each U_i and $e^{iH't}$ can be efficiently implemented, we effectively achieve the implementation of e^{iHt}.

In the following, we first explain the decomposition process and then discuss the implementation of the resulting unitary operations.

Lemma 1. *For a one-sparse Hermitian matrix H with real valued entries, it can be decomposed into the form (we ignore the tensor with identity matrix):*

$$H = M(T \otimes F)M^\dagger \tag{A1}$$

where M is the oracle, T is the swap operator on two registers, and F is diagonal operator, specifically:

$$M|x\rangle|0\rangle|0\rangle|0\rangle = |x\rangle|m(x)\rangle|0\rangle|w(x)\rangle \tag{A2}$$
$$T|x\rangle|y\rangle = |y\rangle|x\rangle \tag{A3}$$
$$F|x\rangle = x|x\rangle \tag{A4}$$

Proof. Suppose the x's column of H has the non-zero element $w(x)$ at position $m(x)$, then we have:

$$H\,|x\rangle = w(x)\,|m(x)\rangle\,.$$

Thus to construct the operator H, is equivalent to mimic such behavior. We can achieve this by the following steps:

1. Apply M to $|x\rangle$ and the ancilla:

$$|x\rangle\,|0\rangle\,|0\rangle\,|0\rangle \xrightarrow{M} |x\rangle\,|m(x)\rangle\,|0\rangle\,|w(x)\rangle\,.$$

2. Apply $T = A^\dagger ZA$:

$$\xrightarrow{T} |m(x)\rangle\,|x\rangle\,|0\rangle\,|w(x)\rangle\,.$$

3. Apply F:

$$\xrightarrow{F} w(x)\,|m(x)\rangle\,|x\rangle\,|0\rangle\,|w(x)\rangle\,.$$

4. Apply $M^\dagger$:

$$\xrightarrow{M^\dagger} w(x)\,|m(x)\rangle\,|x \oplus m(m(x))\rangle\,|0\rangle\,|w(x) \oplus w(m(x))\rangle = w(x)\,|m(x)\rangle\,|0\rangle\,|0\rangle\,|0\rangle\,.$$

Thus H is equivalent to $M(T \otimes F)M^\dagger$. □

Now we are left to implement e^{iTt} and e^{iFt}. Later, we will see that e^{iFt} can be directly implemented. For e^{iTt}, we have to apply unitary conjugation again:

Lemma 2. *For a swap operator T, it can be decomposed into the form:*

$$T = \bigotimes_{l=1}^{n} W^{l,n+l} \bigotimes_{l=1}^{n} E^{l,n+l} \bigotimes_{l=1}^{n} W^{l,n+l} = \tilde{W}\tilde{E}\tilde{W}.$$

where W and E are two qubits operators with the following matrix representation:

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Proof. Notice that T is equivalent to the swap operator on n pairs of qubits, i.e., $T = \bigotimes_{l=1}^{n} S^{l,n+l}$. By diagonalizing S, we obtain WEW, which completes the proof. □

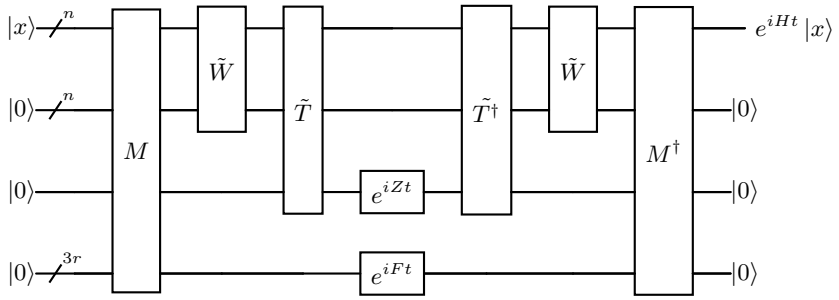


FIG. 2: Quantum circuit for one sparse Hamiltonian simulation.

Again the operator $\tilde{E}$ has to be decomposed, notice that the behavior of $\bigotimes_{l=1}^{n} E^{l,n+l}$ is just compute the parity of $x_i y_i$, and add a phase 1 for even parity, and -1 for odd parity. This can be implemented by using n Toffli-like gates to compute the parity and store in an ancilla qubit, and then apply Z gate to the ancilla. This gives us:

Lemma 3. *For $\tilde{E} = \bigotimes_{l=1}^{n} E^{l,n+l}$, it can be decomposed into the form:*

$$\tilde{E} = \tilde{T_f} Z \tilde{T_f}^\dagger$$

where $\tilde{T_f}$ are set of Toffli-like gates that is used to compute the parity, Z is Z-gate on parity qubit.

By lemma 1, 2, and 3, we have the following theorem:

Theorem 1. *For a one-sparse Hermitian matrix H with real valued entries, e^{iHt} can be implemented by:*

$$e^{iHt} = M\tilde{W}\tilde{T_f}(e^{iZt} \otimes e^{iFt})\tilde{T_f}^\dagger \tilde{W}^\dagger M^\dagger.$$

Proof. By lemma 1, we have:

$$e^{iHt} = e^{iM(T\otimes F)M^\dagger t} = M e^{i(T\otimes F)t} M^\dagger.$$

By lemma 2, we have:

$$e^{i(T\otimes F)t} = e^{i\tilde{W}\tilde{E}\tilde{W}t} = \tilde{W} e^{i\tilde{E}t} \tilde{W}^\dagger.$$

By lemma 3, we have:

$$e^{i\tilde{E}t} = e^{i\tilde{T_f} C_Z \tilde{T_f}^\dagger t} = \tilde{T_f}(C_{e^{iZt}} \otimes e^{iFt})\tilde{T_f}^\dagger.$$

Alltogether, we have:

$$e^{iHt} = M\tilde{W}\tilde{T_f}(C_{e^{iZt}} \otimes e^{iFt})\tilde{T_f}^\dagger \tilde{W} M^\dagger.$$

□

The corresponding quantum circuit for theorem 1 is shown in Figure 2.

In the quantum circuit 2, the oracle M is taken as assumption and we do not count into resource estimation, e^{iZt} is a single qubit rotation gate. Next we'll discuss the implementation of the operators $\tilde{W}$, $\tilde{T}$, and e^{iFt}.

a. Implementation of e^{iFt}

The behavior of e^{iFt} is to apply a phase $e^{i\lambda t}$ to the state $|\lambda\rangle$, where $|\lambda\rangle$ is the binary representation of the value λ. Supposing λ is greater than 0, we have:

$$e^{i\lambda t}|\lambda\rangle = e^{i\sum_{j=1}^{n} 2_j^\lambda t}|\lambda_1 \cdots \lambda_n\rangle \tag{A5}$$

$$= e^{i2_1^\lambda t}|\lambda_1\rangle \otimes e^{i2_2^\lambda t}|\lambda_2\rangle \otimes \cdots \otimes e^{i2_n^\lambda t}|\lambda_n\rangle. \tag{A6}$$

Thus we can implement e^{iFt} by applying phase gate $e^{i2_i^\lambda t}$ on each of the qubit $|\lambda_i\rangle$.

In order to allow negative λ, we have to add another ancilla $|p\rangle$ to indicate the sign, and making the rotation direction of the phase gates controlled by the ancilla. The corresponding circuit is shown in Figure 3.

For each controlled phase gate, we can implement it by using the circuit shown in Figure 4.

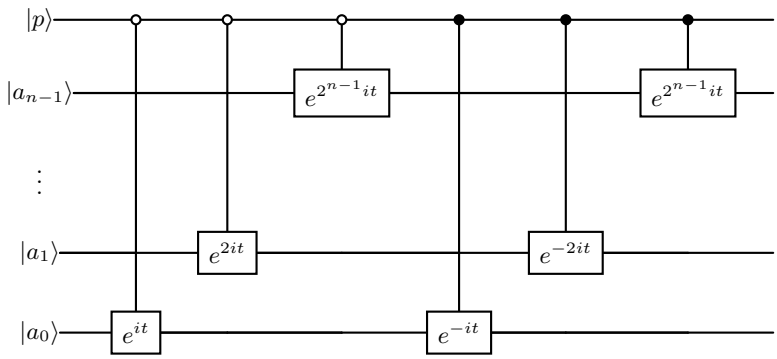


FIG. 3: Quantum circuit for e^{iFt}.

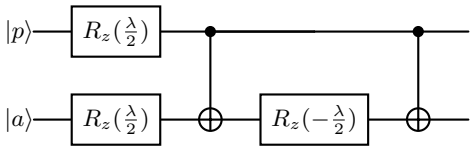


FIG. 4: Controlled phase gate.

b. *Implementation of $\tilde{W}$*

Since $\tilde{W} = \bigotimes_{l=1}^{n} W^{l,n+l}$, $\tilde{W}$ is just apply W on n pairs of qubits, the circuit is shown in Figure 5. Now for each W, since it has the matrix representation 2, we can implement it using the quantum circuit shown in 6.

c. *Implementation of $\tilde{T}$*

The operator $\tilde{T}$ computes the parity of each pair $x_i y_i$ and stores the result in an ancilla qubit. This can be implemented using Toffoli-like gates to compute the parity of each pair $x_i y_i$. The corresponding quantum circuit is shown in Figure 7.

4. Logical resource count

Now we have a concret implementation of one sparse Hamiltonian simulation, we can count the logical resource by adding up the cost of each component. The logical resource is defined as the number of logical qubits and the number of $Clifford + T$ gates.

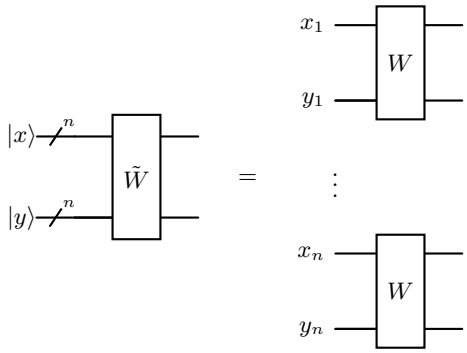


FIG. 5: Quantum circuit implementing the swap operator $\tilde{W}$.

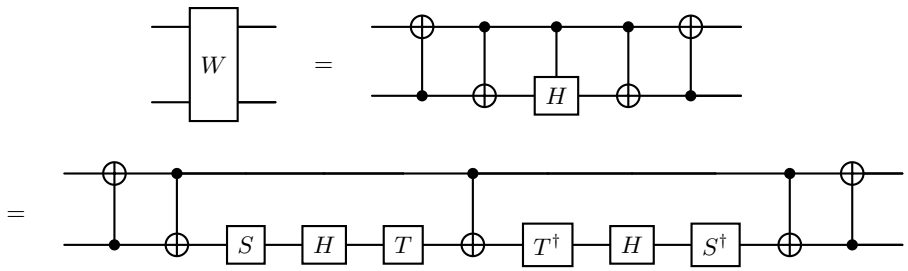


FIG. 6: Quantum circuit implementing the gate W.

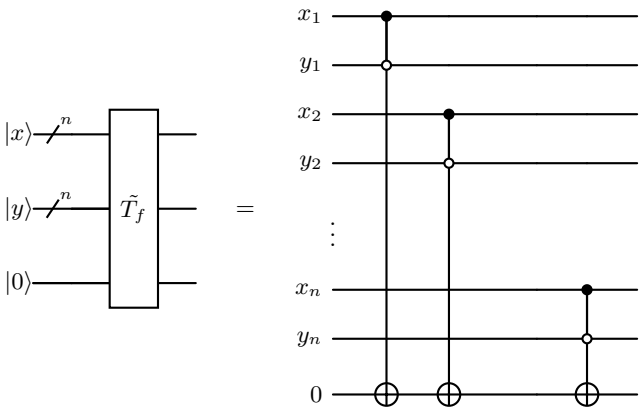


FIG. 7: Quantum circuit implementing the swap operator T.

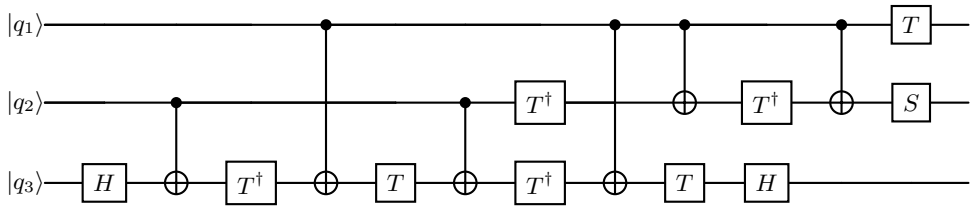


FIG. 8: Quantum circuit implementing the Toffli-like gate.

Component	T gates	$CNOT$	S	H	Explanation
$\tilde{W}$	$4n$	$10n$	$4n$	$4n$	2 $\tilde{W}$ gates, each contains n W gates, and each W gate requires 2 T gates, 5 $CNOT$ gates, 2 S gates and 2 H gates
$\tilde{T}$	$14n$	$12n$	$2n$	$4n$	2 $\tilde{T}$ gates, each contains n Toffoli-like gates, and each Toffoli-like gate requires 7 T gates, 6 $CNOT$ gates, 1 S gate and 2 H gates.
e^{iZt}	15	0	3	3	Assumes, on average, the rotation gates consume 15 T gates, 3 S gates and H gates.
e^{iFt}	$90r$	$4r$	$6r$	$6r$	$2r$ controlled rotation gates, each contains 2 $CNOT$ gate and 3 rotation gates.
Total	$18n + 90r + 15$	$22n + 4r$	$6n + 6r + 3$	$8n + 6r + 3$	Summing up all the components.

TABLE I: Logical T gate count for each component in the algorithm.

Thus we have the following theorem:

Theorem 2. *For a one-sparse Hermitian matrix H with real valued entries, the gate count of the quantum circuit implementing e^{iHt} is given by:*

$$\begin{aligned} T_{count} &= 90r + 18n + 15, \\ CNOT_{count} &= 22n + 4r, \\ S_{count} &= 6n + 6r + 3, \\ H_{count} &= 8n + 6r + 3. \end{aligned}$$

where r is the number of precision bits, n is the number of qubits for input state vector $|x\rangle$.

Appendix B: Scaling analysis

In this section, we derive the exact scaling of the QLSA algorithm. As mentioned in A 2, the bottleneck of the QLSA algorithm lies in the number of one-sparse Hamiltonian simulations (nHS1) we have to perform. This is determined by two factors:

1. The number of one-sparse Hamiltonian simulations required to implement the s-sparse Hamiltonian simulation (the unitary).

2. The number of unitaries (the number of clock qubits, nC) in the quantum phase estimation.

It is crucial to recognize that the QLSA algorithm is essentially an iterative method. To suppress the error to a certain level, we need to perform the subroutine for a specific number of iterations. Thus, analyzing the scaling of the QLSA algorithm is equivalent to bounding the error.

The error of the QLSA algorithm is mainly introduced by two parts: the error introduced by the Trotterization and the error introduced by the Quantum Fourier Transform (QFT). Previous error analyses mainly focused on the latter, and a comprehensive error analysis of both parts has not been provided. Here, we will provide a detailed error analysis of the QLSA algorithm.

The main idea to bound the error of both parts simultaneously is to use an exponential-type error to express the error introduced by Trotterization. We define the **Exponential Type Error** (ETE) as

$$\text{ETE} := \|\tilde{A} - A\|. \tag{B1}$$

This error measure quantifies the deviation in the exponent matrix of the Hamiltonian evolution, which can be viewed as the error of the input. Thus, we can separate the error analysis of Trotterization and QFT, and then add the errors together.

Next, we first derive the exponential-type error of Trotterization, followed by the error of QFT. The derivation is heavily inspired by [12]. Although the paper does not provide an exact error bound for exponential-type error, it provides the main idea of how to derive the error bound. Therefore, it is relatively straightforward to derive it ourselves. We start with:

$$f(t) = e^{iH_1 t} e^{iH_2 t} \cdots e^{iH_n t} \tag{B2}$$

Since f(t) is a unitary operator, we have:

$$f(t) = e^{iH' t} \tag{B3}$$

and we want to bound:

$$\frac{\|H' - H\|}{\|H\|} \tag{B4}$$

Given in the paper, we have:

$$f(t) = \exp_{\mathcal{T}} \left(\int_0^t \mathrm{d}\tau \, (iH + \mathcal{E}(\tau)) \right) \tag{B5}$$

Taking to 1st order in the dysen series expansion of (B5), we have:

$$f(t) = I + iHt + \int_0^t \mathrm{d}\tau\, \mathcal{E}(\tau) + O(t^2) \tag{B6}$$

Also taking 1st order tyler expansion of (B3), we have:

$$f(t) = I + iH't + O(t^2) \tag{B7}$$

Comparing (B6) and (B7), we have:

$$H' = H + \frac{1}{it}\int_0^t \mathrm{d}\tau\, \mathcal{E}(\tau) \tag{B8}$$

Thus:

$$\|H' - H\| = \frac{1}{t}\left\|\int_0^t \mathrm{d}\tau\, \mathcal{E}(\tau)\right\| \tag{B9}$$

In order to get $\mathcal{E}(\tau)$, we differentiate f(t), get:

$$\begin{aligned}\frac{df(t)}{dt} &= (iH_1 + e^{iH_1t}(iH_2)e^{-iH_1t} + ... + e^{iH_1t}...e^{iH_{n-1}t}(iH_n)e^{-iH_{n-1}t}...e^{-iH_1t})f(t) \\ &= (\sum_{j=1}^{n}(\prod_{k=1}^{j-1} e^{iH_k})(iH_jt)(\prod_{k=j-1}^{1} e^{-iH_kt}))f(t) \\ &= (\sum_{j=1}^{n} S_j(t))f(t)\end{aligned} \tag{B10}$$

where:

$$S_j(t) = (\prod_{k=1}^{j-1} e^{iH_kt})(iH_j)(\prod_{k=j-1}^{1} e^{iH_kt}) \tag{B11}$$

Now we have:

$$\frac{df(t)}{dt} = (iH + \mathcal{E}(t))f(t) = (\sum_{j=1}^{n} S_j(t))f(t) \tag{B12}$$

$$\mathcal{E}(t) = \sum_{j=1}^{n} S_j(t) - iH \tag{B13}$$

Next we will separate iHj from $S_j(t)$, the way to achieve this is by recursive expansion. We start with the innermost term. By fundamental theorem of calculus, we have:

$$\begin{aligned}e^{iH_{j-1}t}(iH_j)e^{-iH_{j-1}t} &= iH_j + \int_0^t d\tau (e^{iH_{j-1}\tau}(iH_j)e^{-iH_{j-1}\tau})' \\ &= iH_j + \int_0^t d\tau e^{iH_{j-1}\tau}[iH_{j-1}, iH_j]e^{-iH_{j-1}\tau} \\ &= iH_j + \int_0^t d\tau e^{iH_{j-1}\tau}[H_j, H_{j-1}]e^{-iH_{j-1}\tau}\end{aligned} \tag{B14}$$

For the second layer, we have:

$$
\begin{aligned}
e^{iH_{j-2}}e^{iH_{j-1}t}(iH_j)e^{-iH_{j-1}t}e^{-iH_{j-2}} &= e^{iH_{j-2}t}(iH_j + \int_0^t d\tau e^{iH_{j-1}\tau}[H_j, H_{j-1}]e^{-iH_{j-1}\tau})e^{-iH_{j-2}t} \\
&= e^{iH_{j-2}t}iH_j e^{-iH_{j-2}t} \\
&+ (e^{iH_{j-2}t})\int_0^t d\tau e^{iH_{j-1}\tau}[H_j, H_{j-1}]e^{-iH_{j-1}\tau}(e^{-iHj-2t}) \\
&= iH_j + \int_0^t d\tau e^{iH_{j-2}\tau}[H_j, H_{j-2}]e^{-iH_{j-2}\tau} + \\
&+ (e^{iH_{j-2}t})\int_0^t d\tau e^{iH_{j-1}\tau}[H_j, H_{j-1}]e^{-iH_{j-1}\tau}(e^{-iHj-2t})
\end{aligned} \tag{B15}
$$

Repeat this procedure until all the layers are expanded, we have:

$$
\begin{aligned}
S_j(t) &= (\prod_{k=1}^{j-1} e^{iH_k t})(iH_j)(\prod_{k=j-1}^{1} e^{iH_k t}) \\
&= iH_j + \sum_{k=1}^{j-1}(\prod_{l=1}^{j-k-1} e^{iH_l t})\int_0^t d\tau\, e^{iH_{j-k}\tau}[H_j, H_{j-k}]e^{-iH_{j-k}\tau}(\prod_{l=j-k-1}^{1} e^{-iH_j t})
\end{aligned} \tag{B16}
$$

Then substitute $S_j(t)$ into (B13), we have:

$$
\begin{aligned}
\mathcal{E}(t) &= \sum_{j=1}^{n} S_j(t) - iH \\
&= \sum_{j=1}^{n}\sum_{k=1}^{j-1}(\prod_{l=1}^{j-k-1} e^{iH_l t})\int_0^t d\tau\, e^{iH_{j-k}\tau}[H_j, H_{j-k}]e^{-iH_{j-k}\tau}(\prod_{l=j-k-1}^{1} e^{-iH_j t}) + \sum_{j=1}^{n} iH_j - iH \\
&= \sum_{j=1}^{n}\sum_{k=1}^{j-1}(\prod_{l=1}^{j-k-1} e^{iH_l t})\int_0^t d\tau\, e^{iH_{j-k}\tau}[H_j, H_{j-k}]e^{-iH_{j-k}\tau}(\prod_{l=j-k-1}^{1} e^{-iH_j t})
\end{aligned} \tag{B17}
$$

Now with $\mathcal{E}(t)$ in hand, we can bound (B4) using triangle inequalities:

$$\begin{aligned}
\|H' - H\| &= \frac{1}{t}\left\|\int_0^t \mathrm{d}\tau\, \mathcal{E}(\tau)\right\| \\
&\leq \frac{1}{t}\int_0^t \mathrm{d}\tau\, \|\mathcal{E}(\tau)\| \\
&= \frac{1}{t}\int_0^t \mathrm{d}t_1 \left\|\sum_{j=1}^{n}\sum_{k=1}^{j-1}\Big(\prod_{l=1}^{j-k-1} e^{iH_l t_1}\Big)\int_0^{t_1} \mathrm{d}\tau\, e^{iH_{j-k}\tau}[H_j, H_{j-k}]e^{-iH_{j-k}\tau}\Big(\prod_{l=j-k-1}^{1} e^{-iH_j t_1}\Big)\right\| \\
&\leq \frac{1}{t}\int_0^t \mathrm{d}t_1 \sum_{j=1}^{n}\sum_{k=1}^{j-1}\left\|\Big(\prod_{l=1}^{j-k-1} e^{iH_l t_1}\Big)\int_0^{t_1} \mathrm{d}\tau\, e^{iH_{j-k}\tau}[H_j, H_{j-k}]e^{-iH_{j-k}\tau}\Big(\prod_{l=j-k-1}^{1} e^{-iH_j t_1}\Big)\right\| \\
&= \frac{1}{t}\int_0^t \mathrm{d}t_1 \sum_{j=1}^{n}\sum_{k=1}^{j-1}\left\|\int_0^{t_1} \mathrm{d}\tau\, e^{iH_{j-k}\tau}[H_j, H_{j-k}]e^{-iH_{j-k}\tau}\right\| \\
&\leq \frac{1}{t}\int_0^t \mathrm{d}t_1 \sum_{j=1}^{n}\sum_{k=1}^{j-1}\int_0^{t_1} \mathrm{d}\tau\, \left\| e^{iH_{j-k}\tau}[H_j, H_{j-k}]e^{-iH_{j-k}\tau}\right\| \\
&= \frac{1}{t}\int_0^t \mathrm{d}t_1 \sum_{j=1}^{n}\sum_{k=1}^{j-1}\int_0^{t_1} \mathrm{d}\tau\, \| [H_j, H_{j-k}]\| \\
&\leq \frac{1}{t}\int_0^t \mathrm{d}t_1 \sum_{j=1}^{n}\sum_{k=1}^{j-1}\int_0^{t_1} \mathrm{d}\tau\, 2\, max_m \| H_m\|^2 \\
&= \frac{1}{t}\,\frac{n(n-1)}{2}\,\frac{t^2}{2}\,2\, max_m \| H_m\|^2 \\
&= \frac{n(n-1)t}{2}\, max_m \| H_m\|^2
\end{aligned} \tag{B18}$$

This bound is tight, as we can take example of $n = 2, t = 0.2, H1 = [[1, 0], [0, -1]]$, and $H2 = [[0, 1], [1, 0]]$, the bound and the actual error gives the same value 0.1. However, this bound is significantly above average, the intuition behind this is that in (B13), the error term is a sum of all the commutators, and the commutators can cancel each other, but the bound adds up the norm of all the commutators.

Intuitively this is similar to bounding the sum of random numbers, the bound is the number of random numbers times the maximum of the random numbers.

So the better strategy here is to use probabilistic bound instead, where we tries to bound the error with failure rate below certain threshold.

From the numerical result, we can see that the error does not depend on n. Thus, we can simply do the numerical experiment on small matrix size, and take the upper bound around the edge of distribution, we have:

$$\epsilon = \|\frac{t}{r}\| \tag{B19}$$

Where t is the time step and r is the trotter repetition. Now we have the probabilistic bound on the matrix norm, we can transform it into error on x by applying the following formula:

$$\frac{\|\Delta \mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\kappa \frac{\|\Delta A\|}{\|A\|}}{1 - \kappa \frac{\|\Delta A\|}{\|A\|}} \tag{B20}$$

Since $\|A\| = 1$, and $\|x\| = 1$, we have:

$$\|\Delta \mathbf{x}\| \leq \frac{\kappa \|\Delta A\|}{1 - \kappa \|\Delta A\|} \leq \frac{\kappa t}{r - \kappa t} \tag{B21}$$

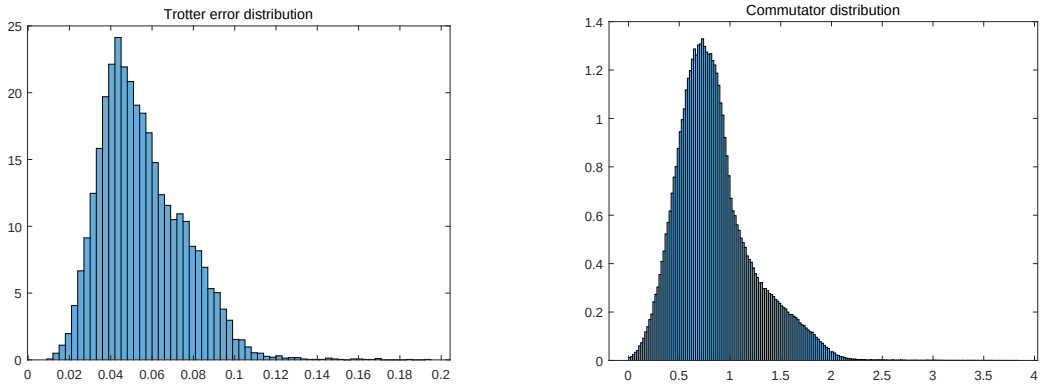


FIG. 9: Trotter error and commutator distribution.

$$\|\Delta x_1\| \leq \sqrt{\frac{20}{3}}\pi\frac{k}{t_0} \tag{B22}$$

Under the constraint:

$$t_0 \leq 2\pi T \Rightarrow \frac{1}{T} \leq 1 - \frac{2\pi\kappa}{t0} \tag{B23}$$

Here, T represents the number of repetitive unitaries, which is 2 to the power of number of clock qubits, and t_0 is the maximum evolution time, given by tT. These constraints ensure that the eigenvalues fall within the range of the QPE estimation. For simplicity, let's assume $t_0 = \pi T$ to meet the first constraint, and later we'll see that the second constraint is satisfied automatically. Then the error in this step is given by:

$$\|\Delta x_2\| \leq \sqrt{\frac{20}{3}}\frac{k}{T} \tag{B24}$$

Now let's combine the error in both steps, by using the triangular inequality:

$$\|\Delta x\| = \|\Delta x_1 + \Delta x_2\| \leq \|\Delta x_1\| + \|\Delta x_2\| \tag{B25}$$

For simplicity we let $\|\Delta x_1 \leq \frac{\epsilon}{2}$ and $\|\Delta x_2 \leq \frac{\epsilon}{2}$, then we have:

$$r \geq \left(\frac{2}{\epsilon} + 1\right)\pi\kappa \approx \frac{2\pi\kappa}{\epsilon} \tag{B26}$$

$$T \geq \sqrt{\frac{80}{3}}\frac{\kappa}{\epsilon} \tag{B27}$$

Finally, we combine those two to get the number of HS1 operations needed:

$$n_{HS1} = Trs \geq \frac{\sqrt{\frac{320}{3}}\pi\kappa^2 s}{\epsilon^2} \tag{B28}$$

Where we assume that there are s number of HS1 per HS, this assumption requires that the oracle needs to implement the seperation of an s-sparse matrix into s number of one sparse matrices, this does not hold in general, but for applications with certain structures this can be done.

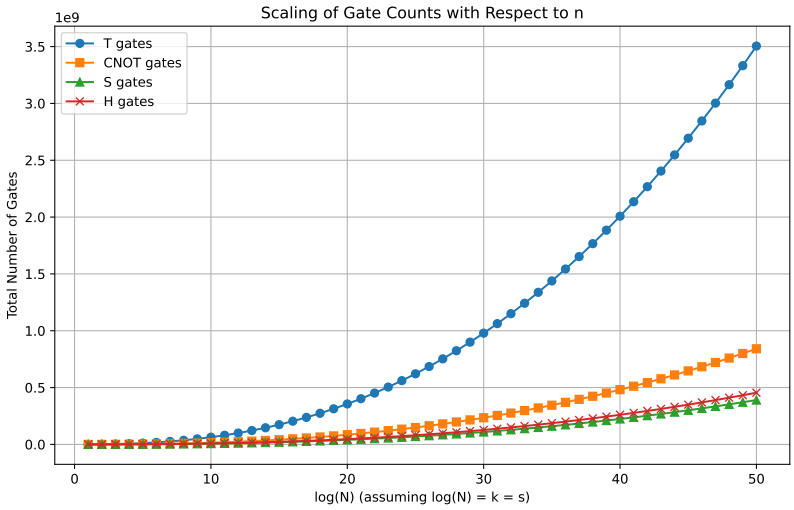


FIG. 10: Scaling of the logical gates with respect to the problem size.

1. Logical resource scaling

In [appendix 1] we derived the scaling of the logical resource of each HS1 operation, and in this section we derived the scaling of number of HS1 operations. By multiply both scaling together, we have the following theorem:

Theorem 3. *The scaling of the logical gates is given by:*

$$n_{gates} = \frac{\sqrt{\frac{320}{3}}\pi\kappa^2 s}{\epsilon^2} \times HS1_{gates}$$

Proof. The proof is straightforward, we just need to multiply the scaling of the logical resource of each HS1 operation with the scaling of the number of HS1 operations. □

Assuming $(\epsilon = 0.1, \kappa = s = \log_2 N)$, we have the scaling of the different gates in plot 10.

Appendix C: Surface code protocol

1. Introduction to surface code

To reliably store and process quantum information over extended periods, active error correction is essential. This is achieved by encoding multiple physical qubits into logical qubits using quantum error-correcting codes [39–41]. Among these, the surface code is not only the most widely used but also one of the most crucial due to its compatibility with the locality constraints of practical quantum hardware, such as superconducting qubits, which only support two-dimensional local operations [42, 43].

However, despite its importance and widespread adoption, the surface code introduces significant computational overhead. The replacement of physical qubits with logical qubits drastically increases space requirements, while the restriction to two-dimensional local operations imposes additional time costs. Arbitrary quantum gates may require multiple time steps instead of executing in a single step, making the choice of surface code schemes—which define parameters such as code distance, logical gate protocols, and resource allocation—critical in determining the actual overhead.

To accurately assess the resource demands of a quantum algorithm, it is essential to analyze the specific surface code scheme under which it is executed. By optimizing these schemes, we can minimize space-time overhead and better evaluate the feasibility of quantum computations within a surface-code-based architecture.

TABLE II: Distillation Protocol Parameters

Protocol	**Number of Magic Tiles**	**Production Time**	**Error Rate Coefficient** (c)
15-1	11	11	$35p^3$
116-12	44	9.27	$4.125p^4$
225-1	176	5.5	$1.5p^7$

2. Key concepts

a. Patch

A *patch* is a two-dimensional regular lattice of entangled physical qubits that serves as the substrate on which logical qubits are defined. Physical qubits within a patch are categorized into two types:

- **Data qubits**: These qubits store quantum information and are measured less frequently, primarily during computational operations.
- **Syndrome qubits**: These qubits interact repeatedly with neighboring data qubits and are frequently measured to detect the presence of errors.

Logical qubits within a patch can perform logical operations (or gates) using a technique known as *lattice surgery*, which is beyond the scope of this discussion. The code distance, denoted as d, determines the error-correcting capability of the patch. A patch of code distance d consists of d^2 physical qubits, with larger values of d yielding higher fidelity computations.

b. Blocks

Blocks are functional units that organize multiple patches according to specific rules. Blocks are categorized as follows:

- **Data blocks**: These accommodate logical data qubits and execute logical operations, including logical gates.
- **Distillation blocks**: These are responsible for generating *magic states*, which are necessary for executing certain non-Clifford gates.

For Clifford gates, computations can be efficiently performed within the data block. However, executing a T-gate requires magic state resources, making it significantly more challenging. The distillation block produces magic states, which are then consumed by the data block to enable T-gate operations. Since both processes rely on complex and time-consuming protocols, T-gates become the primary bottleneck of quantum computation.

c. Protocols

Protocols define how patches within data blocks and distillation blocks are organized and how magic states are produced and consumed. Protocol selection is crucial in optimizing the space-time trade-off of quantum computations.

- **Distillation protocols**: By allocating more patches to a distillation block, magic states can be generated more quickly or with higher fidelity.
- **Data block protocols**: Increasing the number of patches in a data block enables faster consumption of magic states, enhancing computational speed.

A comprehensive set of protocols is described in [13]. The parameters of these protocols are summarized in Table II and III. Later in this work, we optimize the space-time cost of an algorithm with a given T-gate count and physical parameters by selecting appropriate protocols.

TABLE III: Data Protocol Parameters

Protocol	**Magic Consumption Time**	**Number of Tiles per n Qubits**
Compact	9	$1.5n + 3$
Intermediate	5	$2n + 4$
Fast	1	$2n + \sqrt{8n} + 1$

d. Surface code scheme

A *surface code scheme* provides a complete description of a surface code implementation. It is defined by four key parameters:

1. The code distance of the patches.
2. The data block protocol.
3. The distillation protocol.
4. The number of distillation blocks.

These parameters collectively determine the efficiency and reliability of the quantum computation. By carefully selecting and optimizing these elements, it is possible to achieve an optimal balance between resource utilization and computational fidelity.

3. Surface-code-based physical resource estimator

The physical resource estimator is a function that takes the logical resource requirements and quantum computer hardware parameters as inputs and outputs the surface code scheme with the lowest space-time cost. It also provides the corresponding runtime and the number of physical qubits required.

Fundamentally, finding the optimal surface code scheme is a constrained optimization problem. The primary constraint is that both the magic state and the logical qubits must achieve a certain level of fidelity to ensure the final error rate remains below a predefined threshold of 0.01. The objective of the optimization is to minimize the space-time cost, which is defined as the product of runtime and the number of physical qubits. The optimization parameters consist of four key aspects of the surface code scheme: the distillation protocol, the number of distillation blocks, the data block protocol, and the code distance.

We have implemented the optimization procedure outlined in [13] to determine the most efficient surface code scheme. The corresponding pseudocode can be found in 11.

The procedure consists of four steps, which we describe as follows:

a. Step 1: Determine the distillation protocol

A more sophisticated distillation protocol produces magic states with higher fidelity but at the cost of using more patches (and thus more physical qubits). The parameters of several distillation protocols are listed in [reference]. In this step, our goal is to select the most cost-effective distillation protocol that achieves sufficient fidelity to keep the overall computation error rate below 1

The fidelity threshold is defined as the error rate divided by the total number of T gates. We simply choose the least expensive distillation protocol that surpasses this fidelity threshold.

b. Step 2: Construct a minimal setup

In this step, we construct a surface code scheme that occupies the smallest possible space. While this minimal setup is not optimal, it serves as a starting point to determine the code distance. Subsequent optimizations of the data block protocol will build upon this minimal setup.

Algorithm 1 Optimal Surface Code Scheme

Input: (1) Logical qubit count Lq_{count}. (2) Number of T gates T_{count}. (3) Target logical error rate ϵ. (4) Quantum computer error rate p.
Output: Optimized surface code configuration S.

Step 1: Select Distillation Protocol
$F_{\text{threshold}} = \frac{\epsilon}{T_{\text{count}}}$
Let $T_{\text{opt}} = \min(T \mid F(T) < F_{\text{threshold}})$
Step 2: Construct Minimal Setup
$S_{\text{min}} = \{D_{\text{compact}}, T_{\text{opt}}, N_{\text{T}} = 1, d = \text{undefined}\}$
Compute total runtime:

$$t_{\text{runtime}} = T_{\text{count}} \times \max(D_{\text{compact}}.t_{\text{consume}}, T_{\text{opt}}.t_{\text{produce}})$$

Step 3: Determine Code Distance
for $d = d_{\text{min}}$ to d_{max} **do**
    Compute logical error rate:

$$p_L(p, d) = 0.1(100p)^{(d+1)/2}$$

    if $C \cdot d \cdot p_L(10^{-4}, d) < \epsilon$ **then**
        Set $d_{\text{opt}} \leftarrow d$
        break
    end if
end for
Step 4: Optimize Data Block and Distillation Block Count
opt_space_time = space_time(S)
while True **do**
    while $r_{\text{produce}} < r_{\text{consume}}$ **do**
        $N_{\text{distill}} \leftarrow N_{\text{distill}} + 1$
    end while
    while $S[D].t_{\text{produce}} > S[T].t_{\text{consume}}/S[nT]$ **do**
        $S[D] = D'$, where D' is the next larger data block protocol
    end while
end while
Return: Optimized configuration S

FIG. 11: Optimizer Pseudo Code

The minimal setup consists of a compact data block combined with a single distillation block determined in Step 1. The total runtime of this setup is given by the number of T gates multiplied by the clock cycle required to execute each T gate. The latter is determined by the slower operation between magic state production and consumption.

For example, referring to III, the compact data block consumes a magic state every 9 clock cycles, while the 15-to-1 distillation block occupies 11 tiles and outputs a magic state every 11 clock cycles. Since the slower operation dictates the runtime, the effective clock cycle for T gate execution is 11 cycles. If the total T gate count is 10^8, then the algorithm completes in 11×10^8 time steps.

c. Step 3: Determine the code distance

A higher code distance reduces the logical qubit error rate but requires more physical qubits. Denoting the code distance by d, the number of physical qubits per patch is d^2. The logical error rate per logical qubit per code cycle can be approximated as [12]:

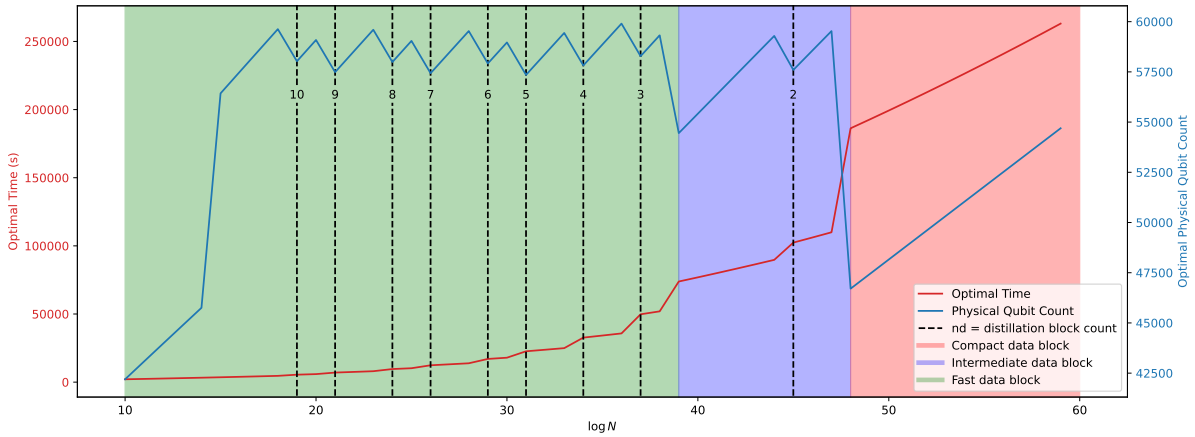


FIG. 12: **Optimizer Result.** This figure shows the optimization results for different protocols, where we used the same setting as 1 except that the physical qubits is restricted to 60000. The red and blue lines represent the optimal time and physical qubit count, respectively. The black dashed lines indicate changes in the distillation block count. The colored background represents different data block protocols.

$$p_L(p,d) = 0.1(100p)^{(d+1)/2}\,. \tag{C1}$$

The code distance must be sufficiently large to suppress the logical error rate such that the total logical error probability for the entire algorithm remains below 1

$$164 \times 11 \times 10^8 \times d \times p_L(10^{-4}, d) < 0.01\,. \tag{C2}$$

d. Step 4: Optimize the data block protocol and distillation block count

Starting from the minimal setup, we identify the performance bottleneck, which typically lies in the speed of magic state production. To accelerate T gate execution, we increase the number of distillation blocks until the magic state production rate surpasses the consumption rate. At this point, we switch to a larger data block protocol to further enhance magic state consumption.

In principle, repeating this process eventually yields a time-optimal surface code scheme, though not necessarily a space-time optimal one. However, for our limited set of distillation and data block protocols, we observe that the time-optimal scheme coincides with the space-time optimal scheme. This suggests that, in the absence of physical qubit constraints, the optimal surface code scheme always employs a "fast block" configuration, with a number of distillation blocks precisely matching the magic state consumption rate.

Thus, in the setting of infinite physicla qubits, what our optimizer does is simply decide the distillation protocol, and code distance, then the data protocol must be fast block, and the number of distillation block must be the smallest number such that the speed of magic state production matches the magic state consumption speed of fast block.

However, more intriguing patterns emerge when the number of physical qubits is constrained. The optimized surface code scheme, as shown in Figure 12, illustrates how the optimal strategy evolves with increasing matrix size. When the matrix size is small, the surface code prioritizes a time-optimal scheme. As the matrix size grows (the sparsity also grows with $\log N$), the scheme gradually reduces the distillation block count, reaching a minimum of three distillation blocks. Beyond this point, further increases in matrix size lead to a transition in data block types, shifting from a standard block to an intermediate block, and eventually to a compact block. Notably, as the data block type changes, the runtime increases significantly.

Algorithm 1 CGNE

Require: $A \in \mathbb{R}^{n\times m}, b \in \mathbb{R}^m, x^0$

$k \leftarrow 0$
$r^0 \leftarrow b - Ax^0$, $z^0 \leftarrow A^T r^0$, $p^0 \leftarrow z^0$
while $\|r^k\| > \epsilon$ **do**
  $w^k \leftarrow Ap^k$
  $\alpha^k \leftarrow \frac{\|z^k\|^2}{\|w^k\|^2}$
  $x^{k+1} \leftarrow x^k + \alpha^k p^k$
  $r^{k+1} \leftarrow r^k - \alpha^k w^k$
  $z^{k+1} \leftarrow A^T r^{k+1}$
  $\beta^k \leftarrow \frac{\|z^{k+1}\|^2}{\|z^k\|^2}$
  $p^{k+1} \leftarrow z^{k+1} + \beta^k p^k$
  $k \leftarrow k + 1$
end while

Ensure: x^k

FIG. 13: CGNE Algorithm

Appendix D: The classical counterparts

1. Conjugate Gradient Method

In this section we introduce the runtime resource estimation of the Conjugate Gradient method.

Definition 1 (LSP)**.** *The Linear System Problem: Find a vector $x \in \mathbb{R}^n$ such that it satisfies equation $Ax = b$ with coefficient matrix $A \in \mathbb{R}^{m\times n}$ and right-hand side (RHS) vector $b \in \mathbb{R}^n$.*

A basic approach for solving an LSP is Gaussian elimination, or LU factorization, with $\mathcal{O}(N^3)$ arithmetic operations. If A is a square symmetric positive semi definite (PSD) matrix, we can also apply Cholesky factorization with $\mathcal{O}(N^3)$ arithmetic operations. The best complexity for an iterative algorithm with respect to N is $\mathcal{O}(Ns\sqrt{\kappa}\log(1/\epsilon))$ arithmetic operations for the Conjugate Gradient (CG) method solving systems with symmetric PSD matrices, where s is the maximum number of non-zero elements in any row or column of A, κ is the condition number of A, and ϵ is the error allowed. If matrix A is just symmetric, one can use the Lanczos algorithm with higher complexity. For an LSP with a general square matrix A, the best iterative method is the GMRES algorithm, which has $\mathcal{O}(n^3)$ worst-case complexity [44]. For problems in the form of $E^TEx = E^T\psi$, known as normal equations, one can use a version of CG methods with complexity $\mathcal{O}(nd\kappa_E \log(1/\epsilon))$, where κ_E is the condition number of matrix E [44]. For a linear system in general form with a non-PSD non-symmetric matrix, one can use the reformulation $A^TAx = A^Tb$ and use a CG method to solve it. Although CG methods for this reformulation have better worst-case complexity than GMRES for the original system, practically GMRES has better performance, especially for large sparse systems with a large condition number [44].

Here, we want to use CGNE (Algorithm 8.5 of [44]) as indicated in Algorithm 13.

As we can see, there is no matrix-matrix product in the CG algorithm. At each iteration, we need to perform two mat-vec products. For each sparse mat-vec product, we require $2Ns$ FLOPs. Additionally, we need three times vector summation with $2N$ FLOPs. Also, $8N$ FLOPs needed for calculating α and β. Total FLOPs needed in each iteration is $4Ns + 6N + 8N$. Thus the dominant cost is for mat-vec products.

Theorem 4. *Starting from $x^0 = 0$, Algorithm 13 reaches to $\frac{\|x^k - x\|}{\|x\|} \leq \epsilon$ after $k \geq \frac{1}{2}\kappa \log\left(\frac{2}{\epsilon}\right)$.*

Proof can be found in [45].

For a s-sparse Hermitian matrix, the total number of the FLOPs are

$$\left(\frac{1}{2}\kappa \log\left(\frac{2}{\epsilon}\right)\right) \times (4Ns + 6N + 8N) \tag{D1}$$

2. Cholesky Decomposition Method

Cholesky decomposition, combined with subsequent Gaussian elimination, can be efficiently parallelized with minimal overhead, particularly for large matrices. This allows us to maintain intermediate matrices in a sparse form, enabling storage within GPU memory.

The algorithm based on Cholesky decomposition consists of the following steps:

1. Given a permutation of rows and columns of the linear equation matrix P and vector b, optimize the process for solving $Px = b$.
2. Construct the necessary data structures to compute and store P.
3. Perform Cholesky decomposition: $P = LL^\top$, where L is a sparse lower triangular matrix.
4. Solve the triangular systems: first, solve $Ly = b$ for y, then solve $L^\top x = y$ for x.

For comparison with quantum algorithm performance evaluations, we exclude GPU data loading and unloading time from our analysis.

Additionally, we conclude that each step of the algorithm is fully parallelizable, with the exception of synchronization overhead, which remains negligible.

We calculate the number of floating-point operations (FLOPs) for each step, where:

- N represents the size of matrix P and vector b.
- s is the maximum number of non-zero elements per row (or column), assuming the matrix is symmetric.

Step 1: Permutation of Rows and Columns

Exploiting matrix symmetry, we traverse each row and swap matrix elements as necessary, leading to:

$$\text{FLOPs} = N \cdot s \cdot (s+1) \tag{D2}$$

Step 2: Data Structure Construction

Assuming the matrix is already stored in an optimal format, the primary operation is generating the index structures, which takes:

$$\text{FLOPs} = N \cdot s \tag{D3}$$

Step 3: Cholesky Decomposition

For each row, we perform factorization computations, resulting in:

$$\text{FLOPs} = N \cdot (1 + s + 2s^2) \tag{D4}$$

Step 4: Triangular Solve

This involves solving two triangular systems, first for y and then for x:

$$\text{FLOPs} = N \cdot (1+s) \cdot 4 \tag{D5}$$

Total FLOPs

Summing all the FLOPs from the above steps:

$$\text{Total FLOPs} = N \cdot s \cdot (s+1) + N \cdot s + N \cdot (1 + s + 2s^2) + N \cdot (1+s) \cdot 4 \tag{D6}$$

$$= N \cdot (3s^2 + 7s + 5) \tag{D7}$$

Since the CD method is fully parallelizable, we can estimate the runtime by simply divide the FLOPS by the supercomputer's execution speed. According to [46], some of the state-of-the-art supercomputer parameters are summerized in IV.

Using these parameters—assuming $k = 10$, $s = \log N$, $\epsilon = 0.01$, and a quantum computer clock cycle of 10 ns—along with a fast block implementation where a logical T operation is executed per clock cycle, we can compute the runtime of the CD method on a supercomputer and compare it with the Quantum Linear Systems Algorithm (QLSA). The results of this comparison are presented in Figure 14.

Our analysis indicates that the CD method exhibits a time complexity of $3Ns^2$, which, in contrast to the CG method (see 2), has a less favorable dependence on s. However, a key advantage of this algorithm is its high degree of parallelizability, enabling efficient execution on supercomputers. Leveraging this capability significantly enhances its performance in terms of runtime. Consequently, the onset of potential quantum advantage may be delayed, as illustrated in Figure 14, where it is projected to emerge at a problem size of $2^{100} \approx 10^{30}$. Nevertheless, since the runtime of QLSA doesn't scaling much with matrix size, the runtime required to reach the potential quantum advantage threshold remains approximately 200 hours.

Name	FLOP/s
Aurora	1.012×10^{18}
El Capitan FP64	2.726×10^{18}
El Capitan FP32	5.453×10^{18}
El Capitan FP16	4.361×10^{19}
El Capitan FP8	8.721×10^{19}
Frontier FP64/32	1.810×10^{18}

TABLE IV: Supercomputer FLOP/s Comparison

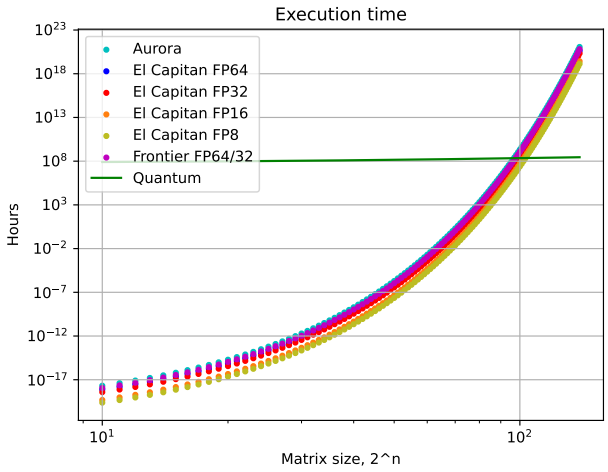


FIG. 14: Runtime comparison of the CD method across different supercomputers.

Appendix E: Quantum energy analysis

The energy consumption of a quantum computer can be estimated as the product of three factors: the scale of the quantum computer (i.e., the number of physical qubits), the energy efficiency of the quantum computer (i.e., power consumption per qubit), and the operating time. This relationship is expressed as:

$$E = n_q \times P_q \times T, \tag{E1}$$

where:

- E is the total energy consumption,
- n_q is the number of physical qubits,
- P_q is the power consumption per physical qubit, and
- T is the total operating time.

The values of n_q and T are determined from prior analysis, leaving P_q as the main parameter to be explained in detail.

a. Power consumption per qubit

As derived in [15], the power consumption per physical qubit, P_q, can be expressed as:

$$P_q = q \left[1 + \phi \left(\frac{1 + \beta n_p^{-1/3}}{\eta_c \mathrm{COP}(T_c)|_c} \right) + \frac{1 - \phi}{\mathrm{FOM}(T_o)} \right], \tag{E2}$$

where:

- q is the computational power per physical qubit,
- ϕ represents the fraction of power used for direct cooling,
- β is the external heat conduction ratio,
- n_p is the number of physical qubits per logical qubit,
- η_c is the efficiency of the cooling system,
- $\mathrm{COP}(T_c)|_c$ is the coefficient of performance (COP) of the cooling system at the chip temperature T_c, and
- $\mathrm{FOM}(T_o)$ is the figure of merit for the secondary cooling system.

The second term in Eq. (E2) accounts for direct heat dissipation, while the third term represents secondary cooling power. Given the early development stage of quantum computers, accurate hardware parameter values for future large-scale fault-tolerant quantum computers remain uncertain, and parameter estimates can vary widely.

b. Simplified estimation of P_q

For a rough estimation of P_q based on current quantum computer parameters, we make the following assumptions:

- The direct heat dissipation power dominates over computational power and secondary cooling power, allowing the omission of the first and third terms in Eq. (E2).
- The volume of the quantum computer is proportional to the number of physical qubits.

Under these assumptions, P_q simplifies to:

$$P_q = \frac{\tilde{q}}{\eta_c \mathrm{COP}(T_c)|_c}, \tag{E3}$$

where $\tilde{q}$ is a parameter to be determined.

c. Determination of $\tilde{q}$

Based on data from [16], IBM's Quantum System Two dilution refrigerator can house 4,158 qubits while consuming 26 kW of power. This yields a per-qubit power consumption of approximately 6.25 W:

$$P_q = \frac{26,000 \text{ W}}{4,158} \approx 6.25 \text{ W}. \tag{E4}$$

For superconducting qubits used by IBM, the coefficient of performance (COP) of the cooling system is approximately 10^{-5}. Substituting into Eq. (E3), we obtain:

$$\tilde{q} = 6.25 \times 10^5. \tag{E5}$$

d. Range of P_q for different architectures

Assuming $\tilde{q}$ remains constant across different architectures, P_q depends primarily on the COP. For a COP ranging from 10^{-5} to 10^{-2}, the power consumption per qubit varies between 6.25 W and 0.0625 W.

1. Conclusion

This analysis provides an approximate estimation of the energy consumption for quantum computers, highlighting the dependence of power consumption on cooling efficiency and quantum computer architecture. Future work will refine these estimates as hardware parameters become more concrete.

Appendix F: Classical energy analysis

The estimation of classical energy consumption follows the formula:

$$E_c = tP_c = \frac{op}{f_c}P_c = op\frac{P_c}{f_c} \tag{F1}$$

where E_c represents the total energy consumption, t denotes the runtime, op is the total number of clock cycles, f_c stands for the clock frequency, and P_c is the power consumption. Consequently, determining the energy efficiency of CPUs, expressed as $\frac{P_c}{f_c}$, is crucial for performance evaluation.

Most mainstream desktop CPUs exhibit power consumption ranging from 50 watts to 400 watts, with clock frequencies spanning from 1 GHz to 5 GHz. Given that power and frequency are positively correlated [47], the power efficiency typically falls within the range of 50 watts per GHz to 80 watts per GHz. In this analysis, we assume a representative value of 50 watts per GHz to estimate the CPU energy consumption of the algorithm.