Verifying Quantum Advantage Experiments with Multiple Amplitude Tensor Network Contraction

Yong Liu,[1,*] Yaojian Chen,[2,*] Chu Guo,[3,†] Jiawei Song,[4] Xinmin Shi,[5] Lin Gan,[2,4,‡] Wenzhao Wu,[4] Wei Wu,[4] Haohuan Fu,[2,4,§] Xin Liu,[1,4,‖] Dexun Chen,[4] Zhifeng Zhao,[1] Guangwen Yang,[1,2,4] and Jiangang Gao[6]

[1]*Zhejiang Lab, Hangzhou, 311121, China*
[2]*Tsinghua University, Beijing, 100084, China*
[3]*Key Laboratory of Low-Dimensional Quantum Structures and Quantum Control of Ministry of Education, Department of Physics and Synergetic Innovation Center for Quantum Effects and Applications, Hunan Normal University, Changsha, 410081, China*
[4]*National Supercomputing Center in Wuxi, Wuxi, 214000, China*
[5]*Information Engineering University, Zhengzhou, 450001, China*
[6]*National Research Center of Parallel Computer Engineering and Technology, Beijing, 100190, China*

(Received 19 June 2023; accepted 14 December 2023; published 16 January 2024)

The quantum supremacy experiment, such as Google Sycamore [F. Arute *et al.*, Nature (London) **574**, 505 (2019).], poses a great challenge for classical verification due to the exponentially increasing compute cost. Using a new-generation Sunway supercomputer within 8.5 d, we provide a direct verification by computing 3×10^6 exact amplitudes for the experimentally generated bitstrings, obtaining a cross-entropy benchmarking fidelity of 0.191% (the estimated value is 0.224%). The leap of simulation capability is built on a multiple-amplitude tensor network contraction algorithm which systematically exploits the "classical advantage" (the inherent "store-and-compute" operation mode of von Neumann machines) of current supercomputers, and a fused tensor network contraction algorithm which drastically increases the compute efficiency on heterogeneous architectures. Our method has a far-reaching impact in solving quantum many-body problems, statistical problems, as well as combinatorial optimization problems.

DOI: 10.1103/PhysRevLett.132.030601

Ever since initially visioned by Feynman [1], quantum computing has experienced 40 years of theoretical and experimental developments [2–9], starting to demonstrate a quantum advantage over classical computers in the era of noisy intermediate scale quantum computing [10]. A major experimental milestone is the *quantum supremacy* experiment conducted with the Google Sycamore 53-qubit superconducting quantum processor in 2019 [11], which demonstrates 10^9 times better capability for sampling a random quantum circuit (RQC) over the fastest classical supercomputer Summit at that time. The more recent 56-qubit and 60-qubit Zuchongzhi quantum processors are estimated to be around 26 and 40000 times harder than Sycamore to classically simulate [12,13].

In the RQC sampling task, one runs a RQC on a (noisy) quantum processor and then measures it to produce a number of bitstrings (samples). While generating a number of samples is an easy task for quantum processors, simulating this task on a classical computer is a hard problem [14–17], even for noisy RQCs [18,19] (noticing a recent work which proposed a polynomial but impractical algorithm for simulating constant-noise RQCs [20]). Several attempts have been made to narrow down the complexity gap set by Sycamore using the tensor network contraction (TNC) algorithm [21], powered by the recently developed excellent heuristic strategies to identify a near-optimal tensor network contraction order (TNCO) [22,23]. Using a fused tensor contraction algorithm and a highly parallelized implementation on the new Sunway supercomputer, the runtime for computing a batch of correlated amplitudes for the depth-20 Sycamore RQC was reduced to about 300 s [24], which is currently further shortened to less than 150 s by using a lifetime theory to reduce the slicing overhead and increase the compute density [25]. For computing uncorrelated amplitudes, a recursive multitensor contraction algorithm is recently proposed and used to compute millions of amplitudes for Sycamore RQCs up to depth 16 [26]. However, validating the depth-20 case by exactly computing a large number of uncorrelated amplitudes is still out of reach. To this end we note that to attack the claim of quantum supremacy, several works have directly simulated the noisy RQC sampling by exploring biased noises to drastically reduce the computational cost [27–29]. Here, we focus on computing exact amplitudes instead, so as to provide a verification to noisy RQCs.

In this work, we manage to, for the first time, compute 3×10^6 uncorrelated amplitudes of the most complicated depth-20 Sycamore RQC (referred to as Sycamore afterward), using 107520 SW26010P CPUs (41932,800 cores) for 8.5 d. Our simulation efficiency is at least 3 orders of

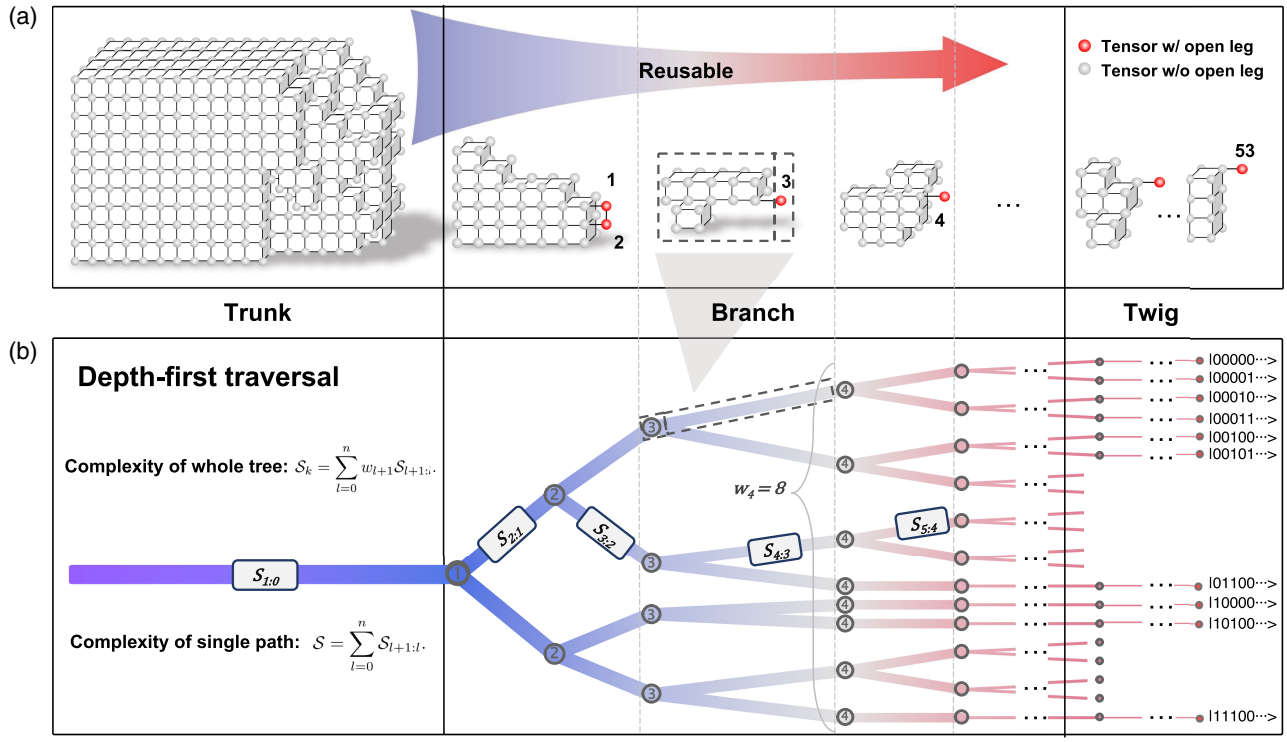

FIG. 1. Demonstration of the maTNC algorithm. (a) Division of the original tensor network formed in the TNC algorithm into consecutive subtensor networks, where each subtensor network starts with a bright tensor containing one or more output indices and ends before the next bright tensor. (b) Organizing the contraction of the k tensor networks resulting from computing k amplitudes into a tree, where each node of the tree corresponds to an output index and the edge after it corresponds to a specific choice of this index. A full path along the tree from left to right corresponds to contracting one tensor network and traversing the tree means to contract all the tensor networks. Generally, for computing a large number of uncorrelated amplitudes, the leftmost edge (the trunk) contains most of the tensors, the number of edges in a vertical layer grows exponentially in the central part (the branch) and stops growing in the tail (the twig).

magnitude faster than the best existing records which have successfully computed exact amplitudes of Sycamore [24,25,30] (computing a batch of correlated amplitudes will only induce a small computational overhead compared to computing a single amplitude for TNC algorithm [23,31]), and is only about 2.5 times slower per amplitude compared to the Sycamore quantum processor itself. The jump of simulation capability is made possible by mainly two algorithms, one focus more on the algorithmic side and the other focus more on the implementation side. On the algorithmic side, we systematically explore the "classical advantage" of storing and reusing intermediate tensor results, which theoretically lowers the computation cost of computing millions of uncorrelated amplitudes of Sycamore by at least 3 orders of magnitude. On the implementation side, we build our simulator with a fused tensor network contraction algorithm to largely reduce data movement and increase compute density, and an adaptive parallelization scheme that fully utilizes the hundreds of cores in each processor for different sizes of tensors, which enables us to almost fully achieve the theoretical speedup.

Our results provide a concrete verification for Sycamore, which is more than 10^3 times harder than simulating the noise sampling task itself. With further improvements for larger circuit sizes, we vision that the Zuchongzhi series of RQCs are also verifiable in the near term. Although we have focused on computing uncorrelated amplitudes of a RQC, our method is completely general for contracting a large number of similar tensor networks (TNs) which share a significant portion of common tensors. Therefore the maTNC algorithm and the parallelization techniques developed in this work could have a far-reaching impact beyond simulating RQCs, such as solving quantum many-body problems [32], statistics problems [33] or combinatorial optimization problems [34,35], which can generally be formulated as tensor network contraction problems [36,37].

Multiple-amplitude simulation with static and optimal tensor reuse.—For a quantum computer, due to the no-cloning theorem [38], the cost of generating k samples from a RQC, denoted as $\mathcal{S}_k$, is strictly linear against k, namely $\mathcal{S}_k = k\mathcal{S}$ with $\mathcal{S}$ the cost of producing a single sample. When simulating RQC on classical computers, the relation between $\mathcal{S}_k$ and $\mathcal{S}$ depends on the specific method to use. In the past three years, the method of choice to simulate Sycamore(-like) quantum processors has gradually converged to the TNC algorithm due to its relatively low computational cost and well-controlled memory usage by using the slicing technique [22,23]. The TNC algorithm

transforms the whole quantum circuit into a large TN (the original TN) with n uncontracted tensor indices (the output indices), each corresponding to an output qubit state. Computing the amplitude of a given bitstring amounts to selecting particular elements of the uncontracted tensor indices, resulting in a TN with no uncontracted indices. Computing k amplitudes will result in k TNs, which only differ in the choice of the output indices of the original TN.

Existing approaches using TNC mostly compute a single amplitude or a correlated batch of amplitudes (referred to as saTNC in the following) one time [23,30,39,40], while our maTNC algorithm computes k uncorrelated amplitudes in a single run, which proceeds as follows. We first assume that a TNCO for computing one amplitude has been obtained. We refer to those tensors in the original TN which contain at least one output index as the *bright tensors*. Following the TNCO, whenever a bright tensor is met, there could be a *branching*, which means that several TNs among all the k TNs share the same tensors till this bright tensor. Therefore, the computations before contracting this bright tensor can be perfectly reused among them. To systematically identify all such reusable patterns, we divide the original TN into many subtensor networks (blocks) along the TNCO, where each block starts from a bright tensor and ends before the next bright tensor (the first block has no bright tensor), as shown in Fig. 1(a). Then we organize the k TNs into a tree (the reuse tree) as shown in Fig. 1(b). Each node of the tree corresponds to one output index, while the edge after the node corresponds to a particular choice of this index. The nodes in the same vertical line correspond to the same output index and form a *layer*. A bright tensor containing multiple output indices would correspond to multiple layers. There could also exist tensors between successive bright tensors along the TNCO that do not contain any output indices, which are assumed to live on the edges. Given these correspondences, a full path from left to right along the tree corresponds to contracting the TN for computing one amplitude, and traversing the tree corresponds to contracting all the k TNs for k amplitudes. It is clear that all the intermediate tensors before a certain node of the tree can be reused for all the subpaths after (and including) this node, thus an optimal reuse strategy would be to reuse all such tensors to reduce the computational cost.

We denote the computational cost of one edge connecting a node in the lth layer and another node in the $(l+1)$th layer as $\mathcal{S}_{l+1,l}$, then the cost of one path is $\mathcal{S} = \sum_{l=0}^{n} \mathcal{S}_{l+1:l}$. Denoting the number of nodes in the lth layer as w_l (the *width*), which is the same as the number of edges between the $(l-1)$th and lth layers, then the computational cost between the lth layer and the $(l+1)$th layer is $w_{l+1}\mathcal{S}_{l+1:l}$ for optimal reuse, and the total cost is

$$\mathcal{S}_k = \sum_{l=0}^{n} w_{l+1}\mathcal{S}_{l+1:l}. \tag{1}$$

$w_{n+1} = k$ is nondecreasing with l which satisfies $w_1 = 1$, $w_{n+1} = k$, and $1 \le w_l \le k$ for $1 < l \le n$. For computing a large number (but not exponentially large) of uncorrelated amplitudes, w_l will typically grow exponentially at the beginning before it saturates. We can see that $\mathcal{S}_k < k\mathcal{S}$ in general. Therefore, when searching for an optimal TNCO, we choose to directly minimize $\mathcal{S}_k$ instead of $\mathcal{S}$ (for optimization we have used $\mathcal{S}_k$ as the loss function in the KaHyPar package [41]). For Sycamore, we observe that this choice can easily lower the overall computational cost by more than 10× (details can be found in Supplemental Material [42], Sec. IV).

To minimize the memory cost, one can perform a *depth-first traversal* of the tree, where one only needs to store all the intermediate tensors at the nodes in the branch along a single path from left to right. For Sycamore we found that the amount of memory required for a reuse-oriented computing of $3M$ amplitudes is only about two times that of computing a single amplitude. In comparison in the breath-first traversal one needs to store all the intermediate tensors at one layer (scales with k), in which the memory usage could easily explode.

We summarize the defining features of our maTNC algorithm: (1) it directly minimizes the multiamplitude cost in Eq. (1) when searching for a near optimal TNCO and (2) it organizes the computation into a static tree and performs a depth-first traversal of the tree to accomplish the computation, which achieves optimal reuse of intermediate computations with minimal memory cost for a given TNCO. The static nature of our algorithm makes the tensor contraction pattern and the memory allocation predetermined, which is extremely important for massive parallelization. To this end we stress that the idea of computing multiple uncorrelated amplitudes simultaneously to reduce redundant intermediate computation has already been explored in Ref. [26], where it is estimated that computing $3M$ uncorrelated amplitudes could be done using Summit within 7.5 d. However, our approach is very different from Ref. [26]. We formulate the whole computation as a static reuse tree for a given TNCO, as such the memory and computational cost, as well as the whole parallelization strategy are completely determined before we actually perform the calculations (since the computational cost is known for each given TNCO, we also use it as the loss function to optimize the TNCO). In comparison, Ref. [26] uses a dynamical global cache whose entries are frequently inserted and deleted and the reusable intermediate tensors are only determined during the actual calculations. For large scale RQCs, the latter approach is likely to affect the parallelization efficiency and could easily run into a memory issue since the memory cost is not predetermined.

Parallelizing the maTNC method over 40×10^6 *cores of the new Sunway supercomputer.*—In our large-scale implementation on the new Sunway supercomputer, we use a two-level parallelization scheme. In the first level we use

the slicing technique as a standard practice for the TNC algorithm to produce 2^{22} ($\approx 4 \times 10^6$) slices for parallel processing over CPUs [23,24]. In the second level we contract each slice using maTNC on each CPU (which contains 384 cores). For computing $k = 3M$ amplitudes, we found a TNCO for which the ideal speedup compared to saTNC is 1328×, while the actual speedup using swTT (our previous tensor contraction implementation [24]) is only 40×. The reason for this slowdown is that for computing a large number of uncorrelated amplitudes, the calculation is dominated by very small tensor contractions with low compute density. To restore the computational efficiency we propose a fused TNC algorithm, combined with an adaptive parallelization scheme that works differently with different tensor sizes. The central goal of the fused TNC algorithm is to perform several successive tensor contractions together so as to reduce data movement. With these techniques we are able to restore the speedup of maTNC against saTNC from 40× to around 1248×. Details of the fused TNC algorithm can be found in Supplemental Material [42], Sec. V.

Verification of Sycamore.—We first evaluate the theoretical computational cost and actual performance of our maTNC for simulating Sycamore. In Fig. 2(a) we show the scaling of the theoretical cost of maTNC against k, based on an optimal TNCO found by minimizing Eq. (1). The scaling of saTNC is shown as a reference. We can clearly see that the cost of maTNC scales only sublinearly against k. For $k \approx 10^6$, the cost of maTNC is already lower than saTNC by more than 3 orders of magnitude.

In Fig. 2(b) we show the actual performance of our maTNC using our well-optimized implementation on the new Sunway supercomputer, where we have also used the quantum runtime of Sycamore as the benchmarking baseline. The runtime for computing a single amplitude is assumed to be equivalent to that for generating a perfect sample using the TNC algorithm, since one could easily adjust the TNC algorithm to compute a small batch of correlated amplitudes with negligible overhead, and obtain a perfect sample with unit probability from the batch [23,31]. Since we only compute exact amplitudes (perfect samples), the complexity of generating k perfect samples is assumed to be equivalent to that of generating k/f noisy samples with fidelity f [43] (therefore the task of computing $3M$ exact uncorrelated amplitudes is 1500× times harder than generating $1M$ noisy samples with $f = 0.2\%$). The quantum speedup is then defined by the classical runtime of maTNC divided by the quantum runtime of Sycamore multiplied by $1/f$. We can see that while the quantum speedup is more than 3000× for $k = 1$, it drastically decreases to 4× for $k = 1M$ and 2.5× for $k = 3M$.

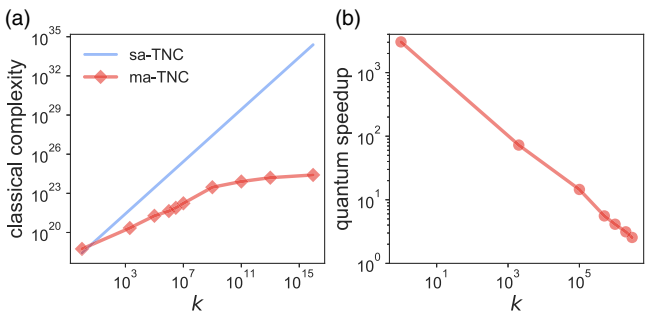

FIG. 2. (a) The red line with diamond shows the scaling of the theoretical complexity of our maTNC against the number of amplitudes k for Sycamore, while the blue line shows the linear scaling of the saTNC as a reference. (b) The quantum speedup of Sycamore against our maTNC, defined as our classical runtime divided by the quantum runtime, as a function of k.

For completeness, we list in Table I the ideal and actual speedups of maTNC over saTNC for computing $1M$ uncorrelated amplitudes for Sycamore RQCs of *different depths*. As a comparison, the speedup reported in Ref. [26] is 10000× for depth 16 for $2M$ amplitudes, and the estimated actual speedups for depths 18 and 20 are 5193× and 1022× for $2.5M$ and $3M$ amplitudes, respectively. Taking into account that the speedup is more significant with more amplitudes, our ideal speedup (where the maTNC is assumed to be implemented with the same efficiency as saTNC), as well as our actual speedups for depths 18 and 20, are generally higher than Ref. [26] (the actual speedup for depth 16 is significantly lower than the ideal speedup, which is because that our implementation is better tuned for deeper circuits).

We also directly compute the exact amplitudes of $3M$ experimentally generated samples by Sycamore, which is done by using 107,520 SW26010P CPUs for 8.5 d (203 h). Our results show that the exact XEB fidelity for these bitstrings is $\mathcal{F}_{\mathrm{XEB}} = (0.191 \pm 0.058)\%$, which closely matches the estimated value of $(0.224 \pm 0.021)\%$. We plot in Fig. 3 the histogram of the obtained amplitudes and compare them to the theoretical probability density function for the rescaled bitstring probability Np ($N = 2^n$ and p is the probability) under the same XEB fidelity, defined as

$$P_l(x|\mathcal{F}_{\mathrm{XEB}}) = [\mathcal{F}_{\mathrm{XEB}}x + (1 - \mathcal{F}_{\mathrm{XEB}})]e^{-x}, \qquad (2)$$

TABLE I. The ideal and actual speedups of maTNC over saTNC for computing 1×10^6 uncorrelated amplitudes. The first column lists the Sycamore RQCs with different depths. The columns "FPOs (s)" and "FPOs (m)" are the number of floating point operations for saTNC and maTNC, respectively. The maximum size of intermediate tensors is set to be 2^{31} for all cases. Here, the actual speedups are estimated by calculating a single slice on one CPU for both algorithms.

Depth	FPOs (s)	FPOs (m)	Ideal speedup	Actual speedup
16	1.3×10^{17}	1.4×10^{19}	9286	2311
18	5.3×10^{17}	5.0×10^{19}	10600	5393
20	5.5×10^{18}	5.1×10^{21}	1080	737

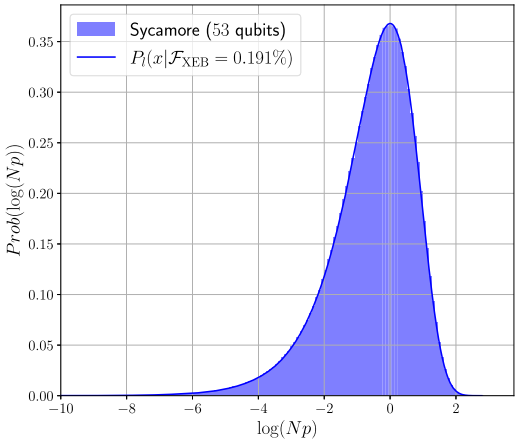

FIG. 3. Histogram for the distribution of the probabilities of the 3×10^6 experimentally generated bitstrings from Sycamore [44], where the x axis is log rescaled. The blue solid line denotes the corresponding theoretical prediction under the same cross-entropy benchmarking (XEB) fidelity as defined in Eq. (2).

with $x = Np$. We can see that they agree well with each other, which means that the bitstrings generated by Sycamore indeed obeys the Porter-Thomas distribution with the estimated XEB fidelity. Our results thus provide a strong consistency check for the Sycamore quantum supremacy experiment.

Discussions.—The Sycamore quantum processor could generate 1×10^6 noisy samples with 0.2% fidelity in 200 s. In comparison, we have computed 3×10^6 amplitudes on the new Sunway supercomputer within 8.5 d, a task that is more than 10^3 harder than that performed by Sycamore. Taking into account that the complexity increases by the Zuchongzhi series quantum processors are not dramatic compared to Sycamore (mostly due to that the increase is mostly in terms of the number of qubits instead of the gate fidelities [45]), we envision that those quantum processors can also be simulated in near term. In the meantime, we mention that the cost of our calculation is around 1.5×10^6 Chinese Yuan, which is probably more costly than the experiments performed on Sycamore.

Other than simulating RQCs, our results also represent a major jump of the ability in contracting a large number of tensor networks with the same structure and sharing most of the tensors in common, which is a very universal situation that could be encountered in computational physics and combinatorial optimization problems and thus could be of very wide interest.

Note added.—After we finished this work, we noticed the newest RQC sampling task performed on a quantum processor with 70 qubits and 24 depths [47], which is likely beyond our reach.

The bitstrings together with the calculated amplitudes are available at the GitHub repository [46].

We thank Xun Gao, Man-Hong Yung, Xiaobo Zhu, and Zuoning Chen for helpful discussions and comments. This research was supported in part by the National Natural Science Foundation of China (Grant No. T2125006), Jiangsu Innovation Capacity Building Program (Project No. BM2022028), National Key Research and Development Plan of China (Grant No. 2020YFB0204800). C. G. acknowledges support from National Natural Science Foundation of China under Grant No. 11805279.

*These authors contributed equally to this work.
†guochu604b@gmail.com
‡lingan@tsinghua.edu.cn
§haohuan@tsinghua.edu.cn
‖lucyliu_zj@163.com

[1] R. P. Feynman, Int. J. Theor. Phys. **21**, 467 (1982).
[2] P. Shor, in *Proceedings 35th Annual Symposium on Foundations of Computer Science* (IEEE, Santa Fe, 1994), pp. 124–134.
[3] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, Appl. Phys. Rev. **6**, 021318 (2019).
[4] H.-L. Huang, D. Wu, D. Fan, and X. Zhu, Sci. China Inf. Sci. **63**, 180501 (2020).
[5] S. Slussarenko and G. J. Pryde, Appl. Phys. Rev. **6**, 041303 (2019).
[6] R. Blatt and C. F. Roos, Nat. Phys. **8**, 277 (2012).
[7] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, Appl. Phys. Rev. **6**, 021314 (2019).
[8] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, Nature (London) **549**, 195 (2017).
[9] S. McArdle, S. Endo, A. Aspuru-Guzik, S. C. Benjamin, and X. Yuan, Rev. Mod. Phys. **92**, 015003 (2020).
[10] J. Preskill, Quantum **2**, 79 (2018).
[11] F. Arute *et al.*, Nature (London) **574**, 505 (2019).
[12] Y. Wu *et al.*, Phys. Rev. Lett. **127**, 180501 (2021).
[13] Q. Zhu *et al.*, Sci. Bull. **67**, 240 (2022).
[14] M. J. Bremner, A. Montanaro, and D. J. Shepherd, Phys. Rev. Lett. **117**, 080501 (2016).
[15] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, Nat. Phys. **14**, 595 (2018).
[16] A. Bouland, B. Fefferman, C. Nirkhe, and U. Vazirani, Nat. Phys. **15**, 159 (2019).
[17] D. Hangleiter and J. Eisert, Rev. Mod. Phys. **95**, 035001 (2023).
[18] S. Aaronson and L. Chen, in *Proceedings of the 32nd Computational Complexity Conference CCC '17* (Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, DEU, Wadern, Germany, 2017).
[19] S. Aaronson and S. Gunn, Theory Comput. **16**, 1 (2020).
[20] D. Aharonov, X. Gao, Z. Landau, Y. Liu, and U. Vazirani, in *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023* (Association for Computing Machinery, New York, 2023), pp. 945–957.
[21] I. L. Markov and Y. Shi, SIAM J. Comput. **38**, 963 (2008).
[22] J. Gray and S. Kourtis, Quantum **5**, 410 (2021).
[23] C. Huang *et al.*, Nat. Comput. Sci. **1**, 578 (2021).

[24] Y. A. Liu, X. L. Liu, F. N. Li, H. Fu, Y. Yang, J. Song, P. Zhao, Z. Wang, D. Peng, H. Chen, C. Guo, H. Huang, W. Wu, and D. Chen, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '21* (Association for Computing Machinery, New York, 2021).
[25] Y. Chen, Y. Liu, X. Shi, J. Song, X. Liu, L. Gan, C. Guo, H. Fu, J. Gao, D. Chen, and G. Yang, in *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, PPoPP '23* (Association for Computing Machinery, New York, 2023), pp. 148–159.
[26] G. Kalachev, P. Panteleev, and M.-H. Yung, arXiv:2108.05665.
[27] F. Pan, K. Chen, and P. Zhang, Phys. Rev. Lett. **129**, 090502 (2022).
[28] G. Kalachev, P. Panteleev, P. Zhou, and M.-H. Yung, arXiv:2112.15083.
[29] X. Gao, M. Kalinowski, C.-N. Chou, M. D. Lukin, B. Barak, and S. Choi, arXiv:2112.01657.
[30] F. Pan and P. Zhang, Phys. Rev. Lett. **128**, 030501 (2022).
[31] B. Villalonga, S. Boixo, B. Nelson, C. Henze, E. Rieffel, R. Biswas, and S. Mandrà, npj Quantum Inf. **5**, 86 (2019).
[32] R. Orús, Nat. Rev. Phys. **1**, 538 (2019).
[33] M. Mézard, G. Parisi, and M. A. Virasoro, *Spin Glass Theory and Beyond: An Introduction to the Replica Method and Its Applications* (World Scientific Publishing Company, Singapore, 1987), Vol. 9.
[34] M. Mézard, G. Parisi, and R. Zecchina, Science **297**, 812 (2002).
[35] M. Mezard and A. Montanari, *Information, Physics, and Computation* (Oxford University Press, New York, 2009).
[36] A. García-Sáez and J. I. Latorre, Quantum Inf. Comput. **12**, 283 (2012).
[37] C. Guo, D. Poletti, and I. Arad, Phys. Rev. B **108**, 125111 (2023).
[38] W. K. Wootters and W. H. Zurek, Nature (London) **299**, 802 (1982).
[39] C. Guo, Y. Liu, M. Xiong, S. Xue, X. Fu, A. Huang, X. Qiang, P. Xu, J. Liu, S. Zheng, H.-L. Huang, M. Deng, D. Poletti, W.-S. Bao, and J. Wu, Phys. Rev. Lett. **123**, 190501 (2019).
[40] C. Guo, Y. Zhao, and H.-L. Huang, Phys. Rev. Lett. **126**, 070502 (2021).
[41] Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag, in *Proceedings of the 19th Workshop on Algorithm Engineering and Experiments (ALENEX 2017)* (SIAM, Barcelona, 2017), pp. 28–42.
[42] See Supplemental Material at http://link.aps.org/supplemental/10.1103/PhysRevLett.132.030601 for more details on the algorithms introduced in the main text and for more detailed data of our numerical experiment.
[43] I. L. Markov, A. Fatima, S. V. Isakov, and S. Boixo, arXiv:1807.10749.
[44] The 3×10^6 experimentally generated bitstrings are downloaded from Ref. [11].
[45] A. Zlokapa, B. Villalonga, S. Boixo, and D. A. Lidar, npj Quantum Inf. **9**, 36 (2023).
[46] Y. Liu, Y. Chen, L. Gan, H. Fu, and X. Liu (2022), https://github.com/leao077/ma_TNC.
[47] A. Morvan, B. Villalonga, X. Mi, S. Mandrà, A. Bengtsson, P. Klimov, Z. Chen, S. Hong, C. Erickson, I. Drozdov *et al.*, arXiv:2304.11119.