Shor's algorithm is possible with as few as 10,000 reconfigurable atomic qubits

Madelyn Cain[1,*,†], Qian Xu[1,2,*,‡], Robbie King[1], Lewis R. B. Picard[1], Harry Levine[1,3],
Manuel Endres[1,2], John Preskill[1,2], Hsin-Yuan Huang[1,2], Dolev Bluvstein[1,2,§]

[1]*Oratomic, Pasadena, California 91125, USA*
[2]*California Institute of Technology, Pasadena, California 91125, USA*
[3]*Department of Physics, University of California, Berkeley, California 94720, USA*
[†]*mcain@oratomic.com,* [‡]*qxu@oratomic.com,* [§]*dbluvstein@oratomic.com*
[*]*These authors contributed equally*
(Dated: March 31, 2026)

Quantum computers have the potential to perform computational tasks beyond the reach of classical machines. A prominent example is Shor's algorithm for integer factorization and discrete logarithms, which is of both fundamental importance and practical relevance to cryptography. However, due to the high overhead of quantum error correction, optimized resource estimates for cryptographically relevant instances of Shor's algorithm require millions of physical qubits. Here, by leveraging advances in high-rate quantum error-correcting codes, efficient logical instruction sets, and circuit design, we show that Shor's algorithm can be executed at cryptographically relevant scales with as few as 10,000 reconfigurable atomic qubits. Increasing the number of physical qubits improves time efficiency by enabling greater parallelism; under plausible assumptions, the runtime for discrete logarithms on the P-256 elliptic curve could be just a few days for a system with 26,000 physical qubits, while the runtime for factoring RSA–2048 integers is one to two orders of magnitude longer. Recent neutral-atom experiments have demonstrated universal fault-tolerant operations below the error-correction threshold, computation on arrays of hundreds of qubits, and trapping arrays with more than 6,000 highly coherent qubits. Although substantial engineering challenges remain, our theoretical analysis indicates that an appropriately designed neutral-atom architecture could support quantum computation at cryptographically relevant scales. More broadly, these results highlight the capability of neutral atoms for fault-tolerant quantum computing with wide-ranging scientific and technological applications.

Quantum computers are believed to be capable of solving certain problems that are intractable for classical machines [1, 2]. A prominent example is Shor's algorithm, which provides superpolynomial speedups over known classical algorithms for integer factorization and discrete logarithms [3, 4], problems that underpin widely used public-key cryptographic systems [5, 6]. Discovered in 1994, this result provided the first evidence that quantum computers could outperform classical computation for problems of both fundamental and practical importance.

The feasibility of large-scale quantum computation was initially questioned due to the fragility of quantum states, but this concern was addressed by pioneering work on quantum error correction in 1995 and 1996 [7–13], which showed that reliable quantum computation is possible provided physical error rates remain below a threshold value. These developments established the theoretical foundations for fault-tolerant quantum computers (FTQCs) capable of executing arbitrarily deep quantum circuits. Shor's algorithm has since served as a flagship benchmark for large-scale quantum computation. Apart from applications to cryptanalysis, FTQCs are expected to enable wide-ranging applications across physics, materials science, quantum chemistry, machine learning, and beyond [14–17].

Despite these theoretical advances, realizing a utility-scale FTQC remains a major challenge. Current resource estimates for large-scale quantum computations—including cryptographically relevant instances of Shor's algorithm—typically require on the order of millions of physical qubits [18]. This scale is driven by the large overhead of quantum error correction, which often necessitates hundreds of physical qubits to encode a single logical qubit [19]. In contrast, today's most advanced quantum processors are limited to at most a few hundred physical qubits. Closing this gap between algorithmic requirements and experimentally accessible hardware is therefore a central obstacle in quantum computing.

The past several decades have seen broad experimental efforts to develop FTQCs across a variety of physical platforms [20–26]. Neutral-atom quantum computers [27] in particular are rapidly emerging as a promising candidate for realizing a FTQC. The first digital quantum circuits were realized in 2022 [28, 29], leveraging parallel reconfigurability of atomic qubits, and shortly thereafter, error-corrected quantum algorithms were performed using up to hundreds of qubits [23, 25, 26, 30]. Despite this rapid progress, today's devices lag far behind the predicted requirements for practically useful quantum computers. Moreover, the comparatively slow clock speeds of neutral-atom systems are thought to potentially limit the practical implementation of deep circuits required for utility-scale processing.

Here we propose and analyze architectures for FTQCs based on reconfigurable neutral-atom systems. Our scheme is based on high-rate error-correcting codes [38–44], which utilize nonlocal connectivity across a code block to protect many logical qubits with minimal overhead. By improving constructions for high-rate codes
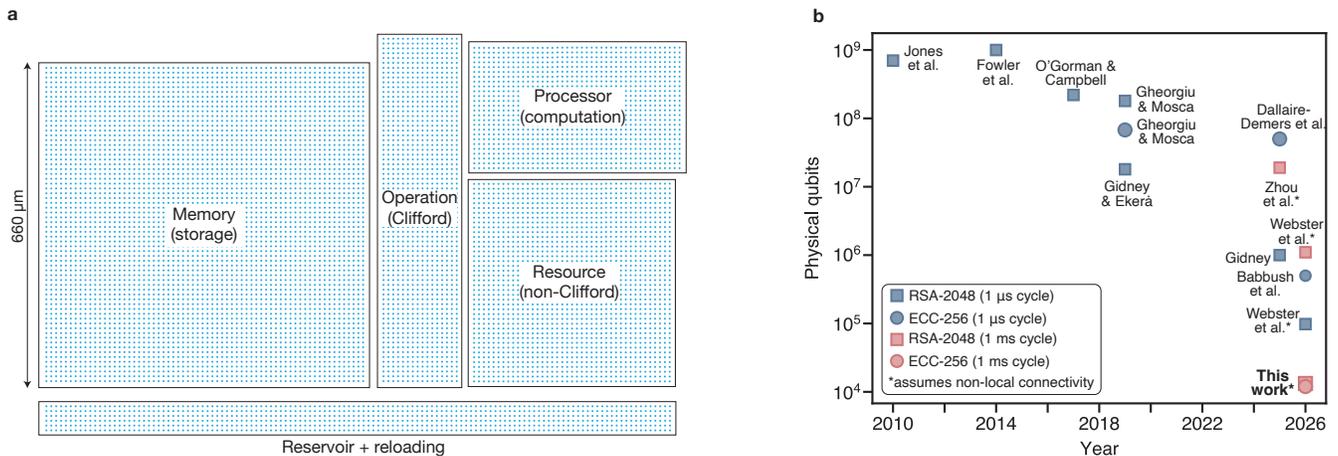
FIG. 1. **Fault-tolerant computation with atomic qubits. a,** Neutral atom processor with four functional zones: a memory zone for storing quantum information, a processor zone for computation, an operation zone for performing Clifford operations, and a resource zone for generating magic states. Also included is a zone for an atomic reservoir and reloading. Each dot represents an atomic data qubit. **b,** Estimated number of physical qubits to run Shor's algorithm versus year of publication for prior resource estimates [18, 19, 31–37] and the current work.

and decoders [45], we encode $\gtrsim 1,000$ logical qubits with $\approx 30\%$ encoding rates at algorithmically relevant logical error rates. This enables qubit requirements to be reduced by an order of magnitude compared to architectures based on small, quasi-local codes with $\sim 4\%$ encoding rates [37, 46, 47], and two orders of magnitude compared to planar surface-code architectures [18]. Leveraging recent advances in logical operations with low space overhead [48–52], we develop a verifiable compilation scheme and project further reductions in time overhead through parallel logical operations [52, 53].

We explore a range of architectures with different numbers of physical qubits and parallelism, and evaluate them using prior state-of-the-art circuits for Shor's algorithm [18, 34, 54, 55], focusing on RSA with 2048-bit keys (RSA–2048) and elliptic-curve cryptography with 256-bit keys (ECC–256). For elliptic-curve discrete logarithms, we explore architectures using approximately 10,000 to 26,000 physical qubits. Applying these architectures to recently developed low-depth circuits [55], we find runtimes varying by roughly two orders of magnitude, with the fastest constructions requiring as few as 10 days, assuming a 1 ms stabilizer measurement cycle time. For the RSA-2048 factoring problem, constructions using 11,000 to 14,000 qubits yield runtimes roughly two orders of magnitude longer due to higher circuit depths [18, 34], while parallelized architectures with $\approx 102,000$ qubits can potentially achieve runtimes of 97 days.

Existing neutral atom systems have demonstrated below-threshold operation, universal fault-tolerant processing on up to 500 qubits [25], and trapping arrays exceeding 6,000 qubits [56]. These advances are complemented by progress in continuous large-scale operation [57–60] and high-fidelity operation across a variety of atomic species [61–64]. While substantial work is needed to integrate these advances into a complete apparatus

and scale system sizes to the required levels, our analysis indicates that appropriately designed neutral-atom architectures could support cryptographically relevant implementations of Shor's algorithm. This finding underscores the importance of ongoing efforts to transition widely-deployed cryptographic systems to post-quantum standards designed to be secure against quantum attacks [65–67]. More broadly, we anticipate that neutral atom processors executing millions of gates on thousands of logical qubits will unlock a wide variety of applications with significant scientific and economic value.

Neutral-atom architecture

Reconfigurable atom arrays offer unique advantages for quantum error correction because qubits can be dynamically rearranged during a computation. In this approach, physical qubits are encoded in long-lived clock states and stored in optical tweezer arrays generated by optical multiplexing devices [27, 68–72]. High-fidelity entangling operations are performed via excitation to Rydberg states [62, 64, 73]. Between gate operations, the qubits are dynamically reconfigured by moving the optical tweezers, enabling massively parallel operation and nonlocal connectivity [28, 74, 75]. Leveraging identical qubits and global parallel control simplifies the implementation of error correction protocols on redundant qubits [30], and the parallel nonlocal connections enable both the use of transversal gates [30, 36, 76, 77] as well as high-rate encodings [26, 30, 44, 78].

Figure 1a presents a conceptual schematic of an atomic quantum computer capable of implementing Shor's algorithm with 11,961 qubits. The computer is divided into four primary functional zones, along with a zone for

an atomic reservoir and reloading. The memory zone stores logical quantum information during the computation. The processor zone stores quantum information undergoing active computation. The operation zone is comprised of ancillary qubits used to perform Clifford logical Pauli product measurements (PPMs), used for reading, writing, and editing of quantum information [49, 51, 52, 79, 80]. Finally, the resource zone generates magic states to elevate Clifford PPMs to universal quantum computation.

Feasible methods exist for operating atomic systems at the scales depicted in Fig. 1a and substantially beyond. Coherent arrays trapping up to 6,100 atomic qubits have already been demonstrated [56] (without yet realizing quantum computation). Laser power overheads can be greatly reduced by holding atoms in optical potentials that are hundreds of times shallower except during imaging, cooling, and transport stages. Additionally, optical tweezer arrays with 360,000 traps have been demonstrated using metasurface phase masks (although without yet trapping atoms) [60].

Furthermore, fault-tolerant architectures have been realized, incorporating simplified methods for removing all relevant entropy sources on systems of hundreds of qubits [25], and including continuous replenishment of lost qubits [57–59]. Operation at $2\times$ below threshold has also been demonstrated, with a path to $10\times$ below threshold [25]. Although high-fidelity entangling operations require high laser intensity and have thus far been realized only on regions of a few hundred qubits [25, 61–64], increasing laser power could directly enable operation on thousands of qubits. Further, the present experimental approach is limited by time-inefficient usage of laser power, where a continuous-wave laser is actively used with only a 0.1% duty cycle (~ 200 ns gate operations separated by idle periods of $\gtrsim 200\,\mu$s). By rastering the beam dynamically to increase the active duty cycle, the number of atoms that can be addressed and entangled at high fidelity could thus be directly increased by three orders of magnitude. Although integrating these capabilities at larger scales requires substantial development effort, they appear to be mutually compatible, such that an appropriately designed architecture could realize the functionality illustrated in Fig. 1a.

Finally, demonstrated speeds with atomic processors are bottlenecked by speeds of readout and motion, which can be $100\,\mu$s to several ms depending on the specific implementations [81]. Recent work, which did not optimize for speed, required several ms per stabilizer measurement cycle [25], albeit in a smaller system than considered in this work. Here we assume a 1 ms stabilizer measurement cycle. Although realizing this in practice may require technological development, we anticipate that it can be achieved by optimizing for speed and leveraging the simple atomic motions inherited from the product structure of the codes we study [25, 30, 44].

Figure 1b plots the resource estimates for cryptographically relevant problem instances as a function of time.

Advances in logical codes, operations, and algorithms have reduced qubit requirements by five orders of magnitude over two decades. Note the differences between RSA–2048, the historical benchmark for Shor's algorithm, and ECC–256. Elliptic curve cryptography is a more modern scheme which provides equivalent classical security, but with significantly smaller key sizes, resulting in reduced quantum complexity. Current state-of-the-art results for planar architectures estimate that RSA–2048 requires 1 million qubits and 1 week [82], whereas recent work finds that ECC–256 requires only half a million qubits and tens of minutes, assuming a $1\,\mu$s cycle time [55]. Here, we explore several architectures with differing resource requirements. We plot two such architectures for RSA–2048 with 13,255 qubits, and ECC–256 with 11,961 qubits (see the end of the resource estimates section for a detailed discussion of our results and prior work).

Codes, logic, and compilation

Reconfigurable atom arrays motivate architectures based on high-rate quantum low-density parity check (qLDPC) codes [83], which leverage nonlocality to densely pack many logical qubits into a single code block [84]. Such codes can substantially reduce the overhead of error correction compared to the millions of qubits required for surface codes [38, 40, 43, 44]. The main challenge is that, unlike the surface code setting, high-rate blocks make it harder to address individual logical qubits and to execute complex algorithms efficiently. Refs. [43, 44], for example, provide concrete protocols for computation with high-rate qLDPC codes, but due to their recent nature, they were not yet highly optimized. Since then, key ingredients for architectures leveraging high-rate codes have improved substantially, particularly decoders [45, 85] and logical operations [49, 51, 52, 79, 80, 86–90].

Figure 2a plots the block error rates of high-rate codes we develop here, leveraging improved quasi-cyclic lifted-product (LP) code [91] constructions with encoding rates of approximately 30%. We analyze three instances from this family with parameters $[[2610, 744, \leq 16]]$, $[[4350, 1224, \leq 20]]$, and $[[5278, 1480, \leq 24]]$, where $[[n, k, d]]$ denotes a code with n physical qubits, k logical qubits, and distance d. We denote these codes as $\mathsf{lp}_{16}^{3,7}$, $\mathsf{lp}_{20}^{3,7}$, and $\mathsf{lp}_{24}^{3,7}$, respectively, where the subscript and superscript correspond to the code distance and seed matrix dimensions respectively; see Appendix A. We use a circuit-level noise model where entangling gates, state preparations, and measurements experience depolarizing noise with error rate p, and decode with a customized belief propagation and localized statistics decoder [45] (Appendix D). At $p = 0.1\%$, the $\mathsf{lp}_{24}^{3,7}$ code achieves extrapolated per-cycle block failure rates (i.e. the probability that any logical qubit fails) of $\approx 10^{-11}$. Further, at low physical error rates, its block er-
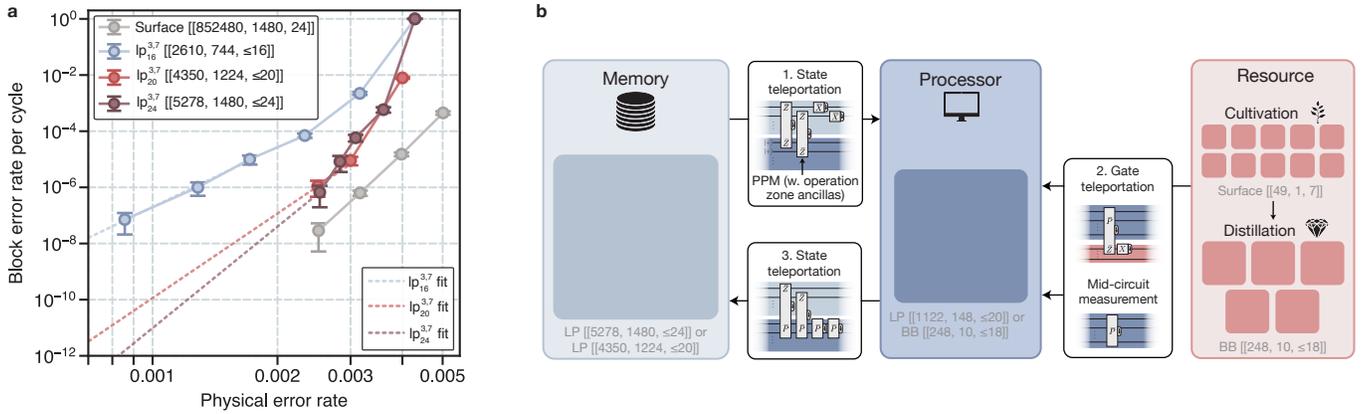
FIG. 2. **Logical code performance and architecture. a,** Block error rates per cycle for several lifted product codes and surface codes. Least-squares power law fits (dashed lines) are used to extrapolate to lower physical error rates p which could not be numerically simulated. The blue fit is of the form $y = ax^b$, where $a = 14.6 \pm 0.7$ and $b = 7.1 \pm 0.4$ are fitted parameters using data from the three smallest physical error rates. Using this same procedure, the fitted values of b for $\mathsf{lp}_{20}^{3,7}$ and $\mathsf{lp}_{24}^{3,7}$ are larger than $d/2$, the theoretical maximum value as $p \to 0$. To be conservative, we therefore fit the form $y = ax^{d/2}$ from the smallest physical error rate (red and purple). **b,** Layout and compilation procedure for the logical architecture. The memory block stores quantum information, which is then teleported to the processor for computation. Sequential PPMs execute mid-circuit measurements and gate teleportation of magic states. Finally, the logical information is teleported back into the memory. Here, $\bar{P}$ denotes an arbitrary logical Pauli operator on the processor code.

ror rates are comparable to surface codes with the same distance and number of logical qubits, but with $161\times$ fewer physical qubits. We note that a hardware-realistic error model with $> 50\%$ of errors due to heralded atom loss and biased Pauli noise could effectively lower p by a factor of two [25, 92–99], further reducing the block error rates. We also anticipate that the current performance is decoder-limited and that further error suppression can be achieved using more advanced decoders [85, 100–102].

With these high-rate codes, one can perform simple resource estimates for a general algorithm. Clifford operations [103] can in principle be implemented directly on a large qLDPC block (e.g., $\mathsf{lp}_{24}^{3,7}$) using code-surgery techniques [48, 49, 87]. These methods use an ancillary system to fault-tolerantly measure an arbitrary logical Pauli operator of a general qLDPC code. Magic $|\bar{T}\rangle$ states may be generated, for example, using surface-code cultivation [50], at an additional space cost of ≈ 500 qubits. The algorithm can then be executed using Pauli-based computation [103], in which Clifford gates are propagated to the end of the circuit, transforming Pauli-product measurements (PPMs) arising from $\bar{T}$-gate teleportation [104] and mid-circuit Pauli measurements into higher-weight logical PPMs, potentially acting on all logical qubits in the code block. In this scheme, each PPM is sequentially implemented via a surgery gadget, such that the total runtime is proportional to the Toffoli count of the algorithm. Each Toffoli gate requires approximately $\tau_{\text{Toff.}} \simeq 4d$ stabilizer measurement cycles, corresponding to 4–7 $|\bar{T}\rangle$-state teleportations per Toffoli gate [105–107].

Crucially, however, the complication in actually executing logic on high-rate codes hides inside of the code surgery implementation, which can present substantial challenges. Such fault-tolerant surgeries can, in principle, be implemented for any qLDPC code, with an ancilla overhead scaling with the block size [49, 87]. However, their practical construction and optimization for large codes, while maintaining low qubit overhead and rigorously benchmarking performance, remain technically challenging and computationally demanding [108]. For this reason, we study below an architecture based on verifiable code surgeries as a theoretically consistent existence proof, which may not be optimal in terms of resource costs. This is achieved by performing computation on smaller high-rate "processor" codes, rather than directly on the large codes in Fig. 2a.

Figure 2b shows the resulting architecture where various qLDPC codes are used for memory, processing, and resource-state generation. An additional operation zone contains ancillary qubits for performing code surgeries. We use either the $\mathsf{lp}_{20}^{3,7}$ or the $\mathsf{lp}_{24}^{3,7}$ code for the memory, depending on the number of logical qubits required for a given algorithm. We consider two possible processor codes: a $[[248, 10, \le 18]]$ bivariate bicycle code [109], denoted bb_{18}, and a $[[1122, 148, \le 20]]$ LP code, denoted $\mathsf{lp}_{20}^{3,5}$ (Appendix A). We refer to architectures with these two choices of the processor code as space-efficient and balanced, respectively, as the bb_{18} code has reduced space cost and logical qubits (and therefore processing power) compared to the $\mathsf{lp}_{20}^{3,5}$ code. Finally, because arithmetic circuits in cryptographic algorithms naturally use non-Clifford Toffoli gates, we generate $|\overline{\text{CCZ}}\rangle$ resource states to directly perform Toffoli gates. We use high-rate $8T$-to-CCZ distillation [51, 110] to distill cultivated $|\bar{T}\rangle$ states [50, 111] into many $|\overline{\text{CCZ}}\rangle$ states hosted in bb_{18} factory codes in parallel. Each resource state has logical

error rate $\lesssim 10^{-10}$ at $p = 0.1\%$, and is generated in time less than a single surgery cycle on average (Appendix C).

Universal computation can be performed on this architecture using the following compilation strategy, detailed in Appendix E and illustrated in Fig. 2b. A circuit, comprised of Clifford and Toffoli gates, is first decomposed into sub-circuits $\{C_i\}$ acting on m_i qubits, where each sub-circuit fits inside the processor block and contains β_i Toffoli gates, γ_i mid-circuit Pauli measurements, and arbitrary Clifford gates. Each C_i is then executed sequentially as follows:

1. *Teleport to processor.* The m_i logical qubits are teleported from the memory code to the processor code using $2m_i$ PPMs: m_i inter-zone measurements between memory and processor, followed by m_i single-qubit measurements on the memory.

2. *Pauli-based computation on the processor.* Clifford gates are absorbed into high-weight PPMs when teleporting in $|\overline{\text{CCZ}}\rangle$ states for Toffoli gates and when implementing mid-circuit measurements.

3. *Teleport to memory.* The m_i qubits are teleported back using another $2m_i$ PPMs: m_i inter-zone measurements (now Clifford-transformed) followed by m_i measurements on the processor.

As described in Appendix B, these PPMs can be implemented using standard code surgery techniques [49, 86, 112] by reconfiguring operation zone ancillary qubits to measure the chosen logical operators. Importantly, the ancilla qubit count scales with the maximum physical weight of the target logical operators. All PPMs in our scheme are of the form $\bar{Z}_i \bar{P}$ or $\bar{X}_j$, where $\bar{Z}_i$ (resp. $\bar{X}_j$) is a single-qubit logical $\bar{Z}$ (resp. $\bar{X}$) operator on the memory or factory code, and $\bar{P}$ is an arbitrary logical operator on the processor code (Fig. 2b). Because the logical operators in the memory and factory codes can be chosen to have low physical weight [113], the ancilla cost therefore scales with the processor code size, rather than with the entire memory block, as would be required if complex operations were performed directly on the memory code. We numerically construct the required surgery gadgets for our architecture and estimate the corresponding ancilla costs, and hence the size of the operation zone, in Appendix B, finding it accounts for only 10%–20% of the total qubit count in both the space-efficient and balanced architectures. Numerical simulations indicate that the logical error rates of the surgery gadgets are within an order of magnitude of the processor and factory code, comparable to the $\mathsf{lp}_{20}^{3,7}$ and $\mathsf{lp}_{24}^{3,7}$ error rates in Fig. 2a. Therefore, we take the memory zone error rate as an approximation for the total architecture error rate.

The time cost of this compilation strategy is as follows. Each PPM is implemented in a single *surgery cycle* consisting of $2d/3$ stabilizer measurement cycles, chosen to minimize the total logical error rate (see Ref. [49] and Appendix B for justification), where d is the processor code distance. Each C_i involves $4m_i + 4\beta_i + \gamma_i$ PPMs,

thus taking $\frac{2d}{3}(4m_i + 4\beta_i + \gamma_i)$ cycles. As the runtime of this compilation strategy scales with the total Toffoli count, we compute the amortized time per Toffoli $\tau_{\text{Toff.}}$ of the circuit. $\tau_{\text{Toff.}}$ is minimized when each C_i contains many Toffoli gates relative to mid-circuit measurements and communication between the memory and processor ($\beta_i \gtrsim m_i, \gamma_i$), i.e. $\tau_{\text{Toff.}} = \frac{2d}{3\beta_i}(4m_i + 4\beta_i + \gamma_i) \geq \frac{8d}{3}$. In practice, however, the exact cost depends on the concrete logical circuit. For Shor's algorithm we therefore estimate $\tau_{\text{Toff.}}$ based on the ripple-carry adder [53, 107, 114] and unary lookup table [115], which we anticipate dominate the Toffoli counts in the circuits we consider [18, 34, 54, 55, 116–120]. As shown in Appendix E, for the balanced architecture we estimate $\tau_{\text{Toff.}} \approx 19 \cdot \frac{2d}{3}$ and $\tau_{\text{Toff.}} \approx 10 \cdot \frac{2d}{3}$ cycles for ECC–256 [55] and RSA–2048 [18, 34], respectively. For the space-efficient architecture, the reduced processor size increases the time to $\tau_{\text{Toff.}} \approx 72 \cdot \frac{2d}{3}$ and $\tau_{\text{Toff.}} \approx 43 \cdot \frac{2d}{3}$ cycles for ECC–256 [55] and RSA–2048 [18, 34], respectively.

The space cost is computed by summing the number of qubits in each zone. We count the data qubits as well as one basis (X or Z) of ancilla qubits for measuring stabilizers by assuming that the X- and Z-stabilizers are measured sequentially. Using the $\mathsf{lp}_{20}^{3,7}$ memory code, the resulting qubit counts are 9,739 and 11,961 for the space-efficient and balanced architectures, respectively, whereas for $\mathsf{lp}_{24}^{3,7}$ memory code, the respective qubit counts are 11,033 and 13,255.

Resource estimates

We now determine the resources required to run Shor's algorithm, focusing on practically relevant benchmarks including Diffie-Hellman (DH), RSA, and ECC. Figure 3a illustrates the required number of logical qubits and Toffoli gate count for prior state-of-the-art circuits [18, 54, 55]. We include both compilations for ECC–256 from Ref. [55], denoted (1) and (2), which offer a tradeoff between logical qubit and Toffoli count. Also plotted are the number of Toffoli gates which can be implemented with 90% total success probability in the balanced architecture, $n_{\text{Toff.}} = \log(0.9)/[\tau_{\text{Toff.}} \log(1 - P_L)]$, where $\tau_{\text{Toff.}} = 19 \cdot \frac{2d}{3}$, and P_L is the extrapolated block error rate per cycle of the memory code. We consider both the $\mathsf{lp}_{20}^{3,7}$ and $\mathsf{lp}_{24}^{3,7}$ memory blocks, which can implement circuits with ≤ 1224 and ≤ 1480 logical qubits, respectively. ECC–256 requires $p = 0.093\%$ with the $\mathsf{lp}_{24}^{3,7}$ memory (compilation (1) can also be implemented with $p = 0.070\%$ using the smaller $\mathsf{lp}_{20}^{3,7}$ memory), whereas RSA–2048 and DH–2048 require slightly reduced physical error rates.

In Figure 3b-c, we plot the number of days required for RSA–2048 and ECC–256, assuming 1 ms per cycle. The time costs vary substantially depending on the logical architectures, circuits, and algorithms. For example, the runtimes of RSA–2048 (Fig. 3b) and ECC–256 (Fig. 3c) vary by nearly two orders of magnitude for the balanced
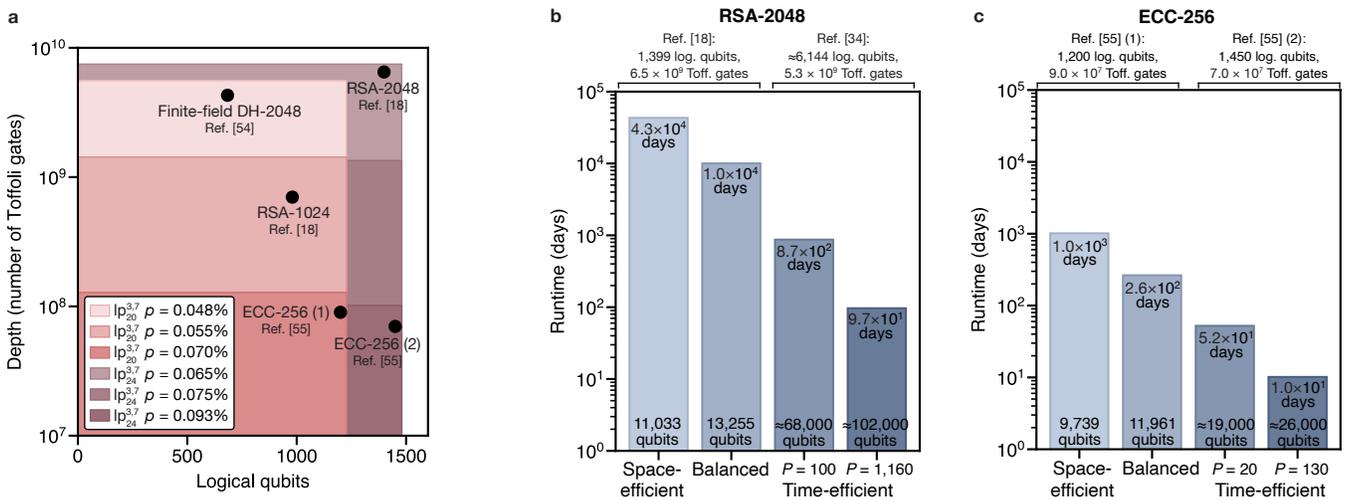
FIG. 3. **Resources required for Shor's algorithm. a,** Number of Toffoli gates and logical qubits required in prior circuits for RSA–1024 and RSA–2048 [18], finite-field DH–2048 [54], and ECC–256 [55]. The colored boxes represent the total number of Toffoli gates that can be executed in the balanced architecture with 90% total success probability, using either the $\mathsf{lp}_{20}^{3,7}$ or $\mathsf{lp}_{24}^{3,7}$ memory. **b,** Runtimes required for RSA–2048 in different architectures (annotations have one significant figure). The space-efficient and balanced architectures use the $\mathsf{lp}_{24}^{3,7}$ memory and the circuit in Ref. [18], whereas the time-efficient architecture uses the circuit in Ref. [34]. We emphasize that the time-efficient runtimes are based on the assumption that up to P Toffoli gates are executed in parallel using high-rate surgery, as motivated by Ref. [52]. **c,** Runtimes required for solving ECC–256, using the compilations in Ref. [55]. The space-efficient and balanced architectures use the $\mathsf{lp}_{20}^{3,7}$ memory.

architecture, with the latter requiring ≈ 264 days. The runtimes for both the space-efficient and balanced architectures are limited by the fact that Toffoli gates and PPMs are executed sequentially. By changing the fault-tolerant schemes for implementing logic and compiling circuits to leverage parallelism, further runtime reductions by nearly two orders of magnitude can potentially be achieved with the time-efficient architecture plotted.

Concretely, in the time-efficient architecture, we consider parallelizing the Toffoli gates in core algorithmic subroutines assuming the use of parallel surgery operations [52, 121, 122]. Such surgery operations have previously been constructed for hundreds of parallel PPMs on related code families, and have demonstrated comparable performance to an idling code block in numerical benchmarks on ≈ 10 PPMs [52]. We estimate the potential speedup by computing the time reduction from replacing n-bit serial quantum ripple-carry adders [53, 107, 114], commonly used in prior circuits [18, 34], with parallelized carry-lookahead adders [123] (here n is the key size). These adders have Toffoli depths (number of maximally parallelized layers of Toffoli gates) of $\approx 1n$–$2n$ and $\approx 4\log(n)$, respectively, where the range corresponds to whether a particular addition is controlled or not. We consider the ECC–256 and RSA–2048 compilations in Refs. [34, 55]. As detailed in Appendix F, we generate and consume $|\overline{\mathrm{CCZ}}\rangle$ states in parallel batches of size P, where the gate teleportation and $\overline{\mathrm{CZ}}$ feedback are directly implemented in three parallel surgery cycles [52, 124]. The estimated speedup increases with P, with projected runtimes of 10 days for ECC–256 ($P = 130$) and 97 days for RSA–2048 ($P = 1,160$).

To realize these time improvements, the number of physical qubits must be increased to generate more $|\overline{\mathrm{CCZ}}\rangle$ states in parallel, and for ancillary logical qubits used in the carry-lookahead adder [123]. We use codes in upcoming work [125], which have 20%–30% encoding rates and maintain the required distance, for storing quantum information, computation, and generating high-rate magic. By assuming that the surgery system scales with the size of codes undergoing parallel operation, in line with prior work [52], we estimate that approximately 26,000 qubits are required for ECC–256 ($P = 130$) and 102,000 qubits are required for RSA–2048 ($P = 1,160$) (see Appendix F for discussion). We emphasize that these estimates are preliminary and can be optimized in future work. Nonetheless, they indicate that the runtime can be substantially reduced by optimizing the algorithm, circuits, and fault-tolerant protocols.

Finally, the advances in this work complement recent progress in the resource estimates plotted in Fig. 1b. Advances in planar surface-code architectures and algorithmic optimizations have already reduced qubit requirements to under one million qubits [55, 82]. Recent progress across hardware platforms has also motivated architectures leveraging nonlocal connectivity to reduce resource costs [36, 37, 46]. Zhou *et al.* [36] compile RSA–2048 with 19 million qubits encoded in surface codes, and uses fast, nonlocal transversal operations to achieve a runtime of ≈ 1 week despite a slower 1 ms cycle time. Architectures based on high-rate codes encoding small ≈ 10 logical qubits have also been proposed, achieving $10\times$ qubit savings over planar architectures while maintaining fixed, quasi-local connectivity and enabling

parallel logical operations across blocks [37, 46]. For example, Webster *et al.* [37] compile Shor's algorithm for RSA–2048 with various resource tradeoffs; we plot two such compilations with 1.1 million qubits and 1 year (1 ms cycle time), and 98,000 qubits and 1 month (1 μs cycle time). Our space-efficient and balanced architectures further reduce qubit counts by one order of magnitude compared to small-block architectures by leveraging the long-range, reconfigurable connectivity of neutral-atom systems. While we expect these architectures have near-optimal qubit counts, our findings indicate that runtimes can be reduced by orders of magnitude by leveraging parallelism [121, 123], suggesting that future efforts should focus on architectural design and reducing algorithm runtime.

Conclusion and outlook

In this work, we propose FTQC architectures based on reconfigurable atom arrays. By optimizing constructions for high-rate codes [38–44], decoders [45], and low-overhead logical operations [48–52], we find that Shor's algorithm can be implemented at cryptographically relevant scales using as few as 10,000 atomic qubits. Leveraging improvements to quantum algorithms [55] and parallel logical operations and circuits [121, 123], our most time-efficient architectures can potentially enable runtimes of 10 days for ECC–256 with $\approx 26,000$ qubits, and 97 days for RSA–2048 with $\approx 102,000$ qubits.

The space and time overheads for these problems are likely to further improve. The space overheads could potentially be reduced by a factor of two beyond our most space-efficient architectures by utilizing higher-rate codes [125]. Substantially larger space reductions, however, would likely require algorithmic compression and improved physical fidelities that permit operation at lower code distances. In contrast, advances in quantum error correction, circuits, and algorithms could potentially reduce runtimes by more than an order of magnitude beyond the time-efficient architecture presented. Each parallel Toffoli layer can potentially be implemented in $\simeq 1$ cycle by using single-shot surgery [79, 90, 126, 127], single-shot code switching [80, 128, 129], or transversal gates [51, 76, 77, 89, 130–132], which could remove the requirement for $d \simeq 20$ cycles per operation. Computation can potentially be further parallelized using space-time tradeoffs [133–135], advanced compilation strategies [37, 136], and improvements to high-rate magic [80, 89]. While we emphasize that such schemes require further innovation and development, they open the possibility of reducing runtime by at least another order of magnitude.

The hardware itself can also be improved along several fronts to reduce runtime. Although systems of $\approx 10,000$ qubits are sufficient to address the relevant problem sizes, known techniques could potentially increase the available space to $\sim 100,000$ qubits. Space-time tradeoffs can then be leveraged to reduce computation time by factors of up to 6–10 $\times$ [133, 134]. Raw qubit speeds can also be improved substantially. Faster readout techniques can reduce measurement times from $\sim 1\,\text{ms}$ to $\sim 1\,\mu s$ [81]. Faster qubit motion can be achieved through architectures that leverage atoms moving at *constant velocity*, thereby avoiding repeated starting and stopping and enabling transport speeds that are potentially hundreds of times faster. These approaches require additional work to incorporate into a complete architecture that removes all forms of entropy [25]. Nevertheless, they indicate that hardware-level speed improvements by several additional orders of magnitude can potentially be achieved, resulting in runtimes on the scale of hours or even minutes.

These findings have significant implications. Although substantial expertise, experimental development effort, and architectural design are required, our theoretical analysis suggests that a neutral atom system capable of implementing Shor's algorithm could be constructed. This conclusion underscores the importance of ongoing efforts to transition widely-deployed cryptographic systems toward post-quantum standards designed to be secure against quantum attacks [65–67]. More broadly, these results position neutral-atom systems as a leading platform for utility-scale quantum computation, with the capacity to drive innovation across science and industry.

[1] Deutsch, D. & Jozsa, R. Rapid solution of problems by quantum computation. *Proc. R. Soc. Lond. A* **439**, 553–558 (1992).

[2] Simon, D. R. On the power of quantum computation. *SIAM J. Comput.* **26**, 1474–1483 (1997).

[3] Shor, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* **26**, 1484–1509 (1997).

[4] Kitaev, A. Y. Quantum measurements and the Abelian Stabilizer Problem. *arXiv preprint arXiv:quant-ph/9511026* (1995).

[5] Rivest, R. L., Shamir, A. & Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **21**, 120–126 (1978).

[6] Koblitz, N. Elliptic curve cryptosystems. *Mathematics of Computation* **48**, 203–209 (1987).

[7] Shor, P. W. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A* **52**, R2493–R2496 (1995).

[8] Calderbank, A. R. & Shor, P. W. Good quantum error-correcting codes exist. *Phys. Rev. A* **54**, 1098–1105 (1996).

[9] Shor, P. W. Fault-tolerant quantum computation. In *Proceedings of 37th conference on foundations of computer science*, 56–65 (IEEE, 1996).

[10] Steane, A. M. Error correcting codes in quantum theory. *Phys. Rev. Lett.* **77**, 793–797 (1996).

[11] Steane, A. M. Simple quantum error-correcting codes. *Phys. Rev. A* **54**, 4741–4751 (1996).

[12] Gottesman, D. Theory of fault-tolerant quantum computation. *Phys. Rev. A* **57**, 127–137 (1998).

[13] Kitaev, A. Y. Fault-tolerant quantum computation by anyons. *Ann. Phys.* **303**, 2–30 (2003).

[14] Dalzell, A. M. *et al.* Quantum algorithms: A survey of applications and end-to-end complexities. *arXiv preprint arXiv:2310.03011* (2023).

[15] Huang, H.-Y., Choi, S., McClean, J. R. & Preskill, J. The vast world of quantum advantage (2025). URL https://arxiv.org/abs/2508.05720. arXiv:2508.05720 [quant-ph].

[16] Babbush, R. *et al.* The grand challenge of quantum applications. *arXiv preprint arXiv:2511.09124* (2025).

[17] Eisert, J. & Preskill, J. Mind the gaps: The fraught road to quantum advantage. *arXiv preprint arXiv:2510.19928* (2025).

[18] Gidney, C. How to factor 2048 bit RSA integers with less than a million noisy qubits (2025). URL https://arxiv.org/abs/2505.15917. arXiv:2505.15917 [quant-ph].

[19] Fowler, A. G., Mariantoni, M., Martinis, J. M. & Cleland, A. N. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A* **86**, 032324 (2012).

[20] Postler, L. *et al.* Demonstration of fault-tolerant universal quantum gate operations. *Nature* **605**, 675–680 (2022).

[21] Wang, Y. *et al.* Fault-tolerant one-bit addition with the smallest interesting color code. *Sci. Adv.* **10** (2024).

[22] Google Quantum AI and Collaborators. Quantum error correction below the surface code threshold. *Nature* **638**, 920–926 (2025).

[23] Reichardt, B. W. *et al.* Logical computation demonstrated with a neutral atom quantum processor (2024). arXiv:2411.11822 [quant-ph].

[24] Ryan-Anderson, C. *et al.* High-fidelity teleportation of a logical qubit using transversal gates and lattice surgery. *Science* **385**, 1327–1331 (2024).

[25] Bluvstein, D. *et al.* A fault-tolerant neutral-atom architecture for universal quantum computation. *Nature* **649**, 39–46 (2026).

[26] Reichardt, B. W. *et al.* Fault-tolerant quantum computation with a neutral atom processor (2025). URL https://arxiv.org/abs/2411.11822. arXiv:2411.11822 [quant-ph].

[27] Saffman, M., Walker, T. G. & Mølmer, K. Quantum information with Rydberg atoms. *Rev. Mod. Phys.* **82**, 2313–2363 (2010).

[28] Bluvstein, D. *et al.* A quantum processor based on coherent transport of entangled atom arrays. *Nature* **604**, 451–456 (2022).

[29] Graham, T. M. *et al.* Multi-qubit entanglement and algorithms on a neutral-atom quantum computer. *Nature* **604**, 457–462 (2022).

[30] Bluvstein, D. *et al.* Logical quantum processor based on reconfigurable atom arrays. *Nature* **626**, 58–65 (2024).

[31] Jones, N. C. *et al.* Layered architecture for quantum computing. *Phys. Rev. X* **2**, 031007 (2012).

[32] O'Gorman, J. & Campbell, E. T. Quantum computation with realistic magic-state factories. *Phys. Rev. A* **95**, 032338 (2017).

[33] Gheorghiu, V. & Mosca, M. Benchmarking the quantum cryptanalysis of symmetric, public-key and hash-based cryptographic schemes (2019). URL https://arxiv.org/abs/1902.02332. arXiv:1902.02332 [quant-ph].

[34] Gidney, C. & Ekerå, M. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum* **5**, 1–31 (2019).

[35] Dallaire-Demers, P.-L., Doyle, W. & Foo, T. Brace for impact: ECDLP challenges for quantum cryptanalysis (2025). URL https://arxiv.org/abs/2508.14011. arXiv:2508.14011 [quant-ph].

[36] Zhou, H. *et al.* Resource Analysis of Low-Overhead Transversal Architectures for Reconfigurable Atom Arrays. In *International Symposium on Computer Architecture* (2025).

[37] Webster, P. *et al.* The Pinnacle Architecture: Reducing the cost of breaking RSA-2048 to 100,000 physical qubits using quantum LDPC codes (2026). URL https://arxiv.org/abs/2602.11457. arXiv:2602.11457 [quant-ph].

[38] Breuckmann, N. P. & Eberhardt, J. N. Quantum Low-Density Parity-Check Codes. *PRX Quantum* **2**, 040101 (2021).

[39] Panteleev, P. & Kalachev, G. Quantum LDPC codes with almost linear minimum distance. *IEEE Trans. Inf. Th.* **68**, 213–229 (2021).

[40] Panteleev, P. & Kalachev, G. Asymptotically good quantum and locally testable classical ldpc codes. In *Proceedings of the 54th annual ACM SIGACT symposium on theory of computing*, 375–388 (2022).

[41] Leverrier, A. & Zémor, G. Quantum tanner codes (2022). URL https://arxiv.org/abs/2202.13641. arXiv:2202.13641 [quant-ph].

[42] Dinur, I., Hsieh, M.-H. H., Lin, T.-C. C. & Vidick, T. Good Quantum LDPC Codes with Linear Time Decoders. *Proceedings of the Annual ACM Symposium on Theory of Computing* 905–918 (2022).

[43] Bravyi, S. *et al.* High-threshold and low-overhead fault-tolerant quantum memory. *Nature* **627**, 778–782 (2024).

[44] Xu, Q. *et al.* Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays. *Nat. Phys.* **20**, 1084–1090 (2024).

[45] Hillmann, T. *et al.* Localized statistics decoding: A parallel decoding algorithm for quantum low-density parity-check codes. *arXiv:2406.18655* (2024).

[46] Yoder, T. J. *et al.* Tour de gross: A modular quantum computer based on bivariate bicycle codes. *arXiv preprint arXiv:2506.03094* (2025).

[47] Yang, W., Chadwick, J., Teo, M. H., Viszlai, J. & Chong, F. Rascql: Reaction-time-limited architecture for space-time-efficient complex qldpc logic. *arXiv preprint arXiv:2602.14273* (2026).

[48] Cohen, L. Z., Kim, I. H., Bartlett, S. D. & Brown, B. J. Low-overhead fault-tolerant quantum computing using long-range connectivity. *Sci. Adv.* **8** (2022).

[49] Cross, A., He, Z., Rall, P. & Yoder, T. Improved QLDPC Surgery: Logical Measurements and Bridging Codes. *arXiv:2407.18393* (2024).

[50] Gidney, C., Shutty, N. & Jones, C. Magic state cultivation: growing T states as cheap as CNOT gates (2024). arXiv:2409.17595 [quant-ph].

[51] Xu, Q. *et al.* Fast and parallelizable logical computation with homological product codes. *Phys. Rev. X* **15**, 021065 (2025).

[52] Zheng, G., Jiang, L. & Xu, Q. High-rate surgery: towards constant-overhead logical operations. *arXiv preprint arXiv:2510.08523* (2025).

[53] Cuccaro, S. A., Draper, T. G., Kutin, S. A. & Moulton, D. P. (2004). arXiv:quant-ph/0410184 [quant-ph].

[54] Chevignard, C., Fouque, P.-A. & Schrottenloher, A. Reducing the number of qubits in quantum factoring. In Tauman Kalai, Y. & Kamara, S. F. (eds.) *Advances in Cryptology – CRYPTO 2025*, 384–415 (Springer Nature Switzerland, Cham, 2025).

[55] Babbush, R. *et al.* Securing elliptic curve cryptocurrencies against quantum vulnerabilities: Resource estimates and mitigations (2026).

[56] Manetsch, H. J. *et al.* A tweezer array with 6,100 highly coherent atomic qubits. *Nature* **647**, 60–67 (2025).

[57] Gyger, F. *et al.* Continuous operation of large-scale atom arrays in optical lattices. *Phys. Rev. Res.* **6**, 033104 (2024).

[58] Norcia, M. A. *et al.* Iterative assembly of ^{171}Yb atom arrays with cavity-enhanced optical lattices. *PRX Quantum* **5**, 030316 (2024).

[59] Chiu, N.-C. *et al.* Continuous operation of a coherent 3,000-qubit system. *Nature* **646**, 1075–1080 (2025).

[60] Holman, A. *et al.* Trapping of single atoms in metasurface optical tweezer arrays. *Nature* **649**, 859–865 (2026).

[61] Evered, S. J. *et al.* High-fidelity parallel entangling gates on a neutral-atom quantum computer. *Nature* **622**, 268–272 (2023).

[62] Ma, S. *et al.* High-fidelity gates and mid-circuit erasure conversion in an atomic qubit. *Nature* **622**, 279–284 (2023).

[63] Lis, J. W. *et al.* Midcircuit Operations Using the omg Architecture in Neutral Atom Arrays. *Physical Review X* **13** (2023).

[64] Tsai, R. B.-S., Sun, X., Shaw, A. L., Finkelstein, R. & Endres, M. Benchmarking and Fidelity Response Theory of High-Fidelity Rydberg Entangling Gates. *PRX Quantum* **6**, 010331 (2025).

[65] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. Tech. Rep. Federal Information Processing Standards Publications (FIPS) 203, U.S. Department of Commerce, Washington, D.C. (2024).

[66] National Institute of Standards and Technology. Module-lattice-based digital signature standard. Tech. Rep. Federal Information Processing Standards Publications (FIPS) 204, U.S. Department of Commerce, Washington, D.C. (2024).

[67] National Institute of Standards and Technology. Stateless hash-based digital signature standard. Tech. Rep. Federal Information Processing Standards Publications (FIPS) 205, U.S. Department of Commerce, Washington, D.C. (2024).

[68] Barredo, D., De Léséleuc, S., Lienhard, V., Lahaye, T. & Browaeys, A. An atom-by-atom assembler of defect-free arbitrary two-dimensional atomic arrays. *Science* **354**, 1021–1023 (2016).

[69] Endres, M. *et al.* Atom-by-atom assembly of defect-free one-dimensional cold atom arrays. *Science* **354**, 1024–1027 (2016).

[70] Ma, S. *et al.* Universal gate operations on nuclear spin qubits in an optical tweezer array of ^{171}Yb atoms. *Phys. Rev. X* **12**, 021028 (2022).

[71] Jenkins, A., Lis, J. W., Senoo, A., McGrew, W. F. & Kaufman, A. M. Ytterbium nuclear-spin qubits in an optical tweezer array. *Phys. Rev. X* **12**, 021027 (2022).

[72] Finkelstein, R. *et al.* Universal quantum operations and ancilla-based read-out for tweezer clocks. *Nature* **634**, 321–327 (2024).

[73] Isenhower, L. *et al.* Demonstration of a Neutral Atom Controlled-NOT Quantum Gate. *Phys. Rev. Lett.* **104**, 010503 (2010).

[74] Beugnon, J. *et al.* Two-dimensional transport and transfer of a single atomic qubit in optical tweezers. *Nature Physics* **3**, 696–699 (2007).

[75] Schlosser, M., Tichelmann, S., Kruse, J. & Birkl, G. Scalable architecture for quantum information processing with atoms in optical micro-structures. *Quantum Information Processing* **10**, 907–924 (2011).

[76] Cain, M. *et al.* Correlated Decoding of Logical Algorithms with Transversal Gates. *Phys. Rev. Lett.* **133**, 240602 (2024).

[77] Zhou, H. *et al.* Algorithmic fault tolerance for fast quantum computing (2024). arXiv:2406.17653 [quant-ph].

[78] Zhang, B. *et al.* Leveraging erasure errors in logical qubits with metastable ^{171}yb atoms (2025). URL https://arxiv.org/abs/2506.13724. arXiv:2506.13724 [quant-ph].

[79] Cowtan, A., He, Z., Williamson, D. J. & Yoder, T. J. Fast and fault-tolerant logical measurements: Auxiliary hypergraphs and transversal surgery (2025). URL https://arxiv.org/abs/2510.14895. arXiv:2510.14895 [quant-ph].

[80] Li, C., Preskill, J. & Xu, Q. Transversal dimension jump for product qldpc codes (2025). URL https://arxiv.org/abs/2510.07269. arXiv:2510.07269 [quant-ph].

[81] Muzi Falconi, A. *et al.* Microsecond-scale high-survival and number-resolved detection of ytterbium atom arrays. *Phys. Rev. Lett.* **135**, 203402 (2025).

[82] Gidney, C. How to factor 2048 bit rsa integers with less than a million noisy qubits. *arXiv preprint arXiv:2505.15917* (2025).

[83] Breuckmann, N. P. & Eberhardt, J. N. Quantum low-density parity-check codes. *PRX Quantum* **2**, 040101 (2021).

[84] Bravyi, S., Poulin, D. & Terhal, B. Tradeoffs for Reliable Quantum Information Storage in 2D Systems. *Phys. Rev. Lett.* **104**, 050503 (2010).

[85] Müller, T. *et al.* Improved belief propagation is sufficient for real-time decoding of quantum memory. *arXiv preprint arXiv:2506.01779* (2025).

[86] Williamson, D. J. & Yoder, T. J. Low-overhead fault-tolerant quantum computation by gauging logical operators. *arXiv preprint arXiv:2410.02213* (2024).

[87] He, Z., Cowtan, A., Williamson, D. J. & Yoder, T. J. Extractors: Qldpc architectures for efficient pauli-based computation. *arXiv preprint arXiv:2503.10390* (2025).

[88] Malcolm, A. J. *et al.* Computing efficiently in qldpc codes. *arXiv preprint arXiv:2502.07150* (2025).

[89] Menon, V. *et al.* Magic tricycles: Efficient magic state generation with finite block-length quantum ldpc codes (2025). URL https://arxiv.org/abs/2508.10714. arXiv:2508.10714 [quant-ph].

[90] Xu, Q. *et al.* Batched high-rate logical operations for quantum ldpc codes. *arXiv preprint arXiv:2510.06159* (2025).

[91] Panteleev, P. & Kalachev, G. Degenerate Quantum LDPC Codes With Good Finite Length Performance. *Quantum* **5**, 585 (2019).

[92] Wu, Y., Kolkowitz, S., Puri, S. & Thompson, J. D. Erasure conversion for fault-tolerant quantum computing in alkaline earth Rydberg atom arrays. *Nat. Comm.* **13**, 1–7 (2022).

[93] Sahay, K., Jin, J., Claes, J., Thompson, J. D. & Puri, S. High threshold codes for neutral atom qubits with biased erasure errors. *arXiv preprint arXiv:2302.03063* (2023).

[94] Kubica, A. *et al.* Erasure qubits: Overcoming the T_1 limit in superconducting circuits. *Phys. Rev. X* **13**, 041022 (2023).

[95] Yu, C.-C. *et al.* Processing and decoding Rydberg decay error with MBQC (2025). arXiv:2411.04664 [quant-ph].

[96] Chang, K. *et al.* Surface Code with Imperfect Erasure Checks (2024). arXiv:2408.00842 [quant-ph].

[97] Perrin, H., Jandura, S. & Pupillo, G. Quantum error correction resilient against atom loss (2025). arXiv:2412.07841 [quant-ph].

[98] Baranes, G. *et al.* Leveraging atom loss errors in fault tolerant quantum algorithms (2025). arXiv:2502.20558 [quant-ph].

[99] Gu, S., Retzker, A. & Kubica, A. Fault-tolerant quantum architectures based on erasure qubits. *Phys. Rev. Res.* **7**, 013249 (2025).

[100] Senior, A. W. *et al.* A scalable and real-time neural decoder for topological quantum codes (2026). URL https://arxiv.org/abs/2512.07737. arXiv:2512.07737 [quant-ph].

[101] Maan, A. S., Garcia Herrero, F. M., Paler, A. & Savin, V. Decoding correlated errors in quantum ldpc codes. *Nature Communications* (2026).

[102] Ataides, J. P. B., Gu, A., Yelin, S. F. & Lukin, M. D. Neural decoders for universal quantum algorithms (2025). URL https://arxiv.org/abs/2509.11370. arXiv:2509.11370 [quant-ph].

[103] Bravyi, S., Smith, G. & Smolin, J. A. Trading classical and quantum computational resources. *Physical Review X* **6**, 021043 (2016).

[104] A $\overline{T}$ gate on the memory code can be implemented using a two-qubit $\bar{Z}\bar{Z}$ measurement between the memory and the surface code hosting the $|\overline{T}\rangle$ state [137].

[105] Jones, C. Low-overhead constructions for the fault-tolerant toffoli gate. *Phys. Rev. A* **87**, 022328 (2013).

[106] Amy, M., Maslov, D., Mosca, M. & Roetteler, M. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *Trans. Comp.-Aided Des. Integ. Cir. Sys.* **32**, 818–830 (2013).

[107] Gidney, C. Halving the cost of quantum addition. *Quantum* **2**, 74 (2018).

[108] State-of-the-art, low-overhead constructions rely on randomized numerical subroutines that extensively estimate the fault distance of the protocol [49, 52], a procedure that becomes computationally prohibitive for large code blocks.

[109] Liang, Z., Liu, K., Song, H. & Chen, Y.-A. Generalized toric codes on twisted tori for quantum error correction. *arXiv preprint arXiv:2503.03827* (2025).

[110] Litinski, D. Magic State Distillation: Not as Costly as You Think. *Quantum* **3**, 205 (2019).

[111] Sahay, K. *et al.* Fold-transversal surface code cultivation (2025). URL https://arxiv.org/abs/2509.05212. arXiv:2509.05212 [quant-ph].

[112] Swaroop, E., Jochym-O'Connor, T. & Yoder, T. J. Universal adapters between quantum ldpc codes. *arXiv preprint arXiv:2410.03628* (2024).

[113] Zheng, H., Zheng, G., Jiang, L. & Xu, Q. In preparation (2026).

[114] Vedral, V., Barenco, A. & Ekert, A. Quantum networks for elementary arithmetic operations. *Physical Review A* **54**, 147 (1996).

[115] Babbush, R. *et al.* Encoding electronic spectra in quantum circuits with linear t complexity. *Phys. Rev. X* **8**, 041015 (2018).

[116] Proos, J. & Zalka, C. Shor's discrete logarithm quantum algorithm for elliptic curves. *arXiv preprint quant-ph/0301141* (2003).

[117] Roetteler, M., Naehrig, M., Svore, K. M. & Lauter, K. Quantum resource estimates for computing elliptic curve discrete logarithms. In *International Conference on the Theory and Application of Cryptology and Information Security*, 241–270 (Springer, 2017).

[118] Häner, T., Jaques, S., Naehrig, M., Roetteler, M. & Soeken, M. Improved quantum circuits for elliptic curve discrete logarithms. In *International conference on post-quantum cryptography*, 425–444 (Springer, 2020).

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

[120] Gouzien, E., Ruiz, D., Le Régent, F.-M., Guillaud, J. & Sangouard, N. Performance analysis of a repetition cat code architecture: Computing 256-bit elliptic curve logarithm in 9 hours with 126 133 cat qubits. *Phys. Rev. Lett.* **131**, 040602 (2023).

[121] Zhang, G. & Li, Y. Time-efficient logical operations on quantum low-density parity check codes. *Physical*

Review Letters **134**, 070602 (2025).

[122] Cowtan, A., He, Z., Williamson, D. J. & Yoder, T. J. Parallel logical measurements via quantum code surgery. *arXiv preprint arXiv:2503.05003* (2025).

[123] Draper, T. G., Kutin, S. A., Rains, E. M. & Svore, K. M. A logarithmic-depth quantum carry-lookahead adder. *Quantum Inf. Comput.* **6**, 351–369 (2004).

[124] Horsman, C., Fowler, A. G., Devitt, S. & Meter, R. V. Surface code quantum computing by lattice surgery. *New Journal of Physics* **14**, 123011 (2012).

[125] Caltech & Oratomic. Atlas of atomic fault-tolerant quantum memory (2026). In preparation.

[126] Baspin, N., Berent, L. & Cohen, L. Z. Fast surgery for quantum ldpc codes. *arXiv preprint arXiv:2510.04521* (2025).

[127] Chang, K., He, Z., Yoder, T. J., Zhu, G. & Jochym-O'Connor, T. Constant-time surgery on 2d hypergraph product codes with near-constant space overhead. *arXiv preprint arXiv:2603.02157* (2026).

[128] Tan, S. J. S., Hong, Y., Lin, T.-C., Gullans, M. J. & Hsieh, M.-H. Single-shot universality in quantum ldpc codes via code-switching (2025). URL https://arxiv.org/abs/2510.08552. arXiv:2510.08552 [quant-ph].

[129] Golowich, L., Chang, K. & Zhu, G. Constant-overhead addressable gates via single-shot code switching. *arXiv preprint arXiv:2510.06760* (2025).

[130] Bombín, H. Gauge Color Codes: Optimal Transversal Gates and Gauge Fixing in Topological Stabilizer Codes. *New Journal of Physics* **17** (2013).

[131] Brown, B. J., Nickerson, N. H. & Browne, D. E. Fault-tolerant error correction with the gauge color code. *Nature Communications* **7**, 12302 (2016).

[132] Jacob, A., McLauchlan, C. & Browne, D. E. Single-shot decoding and fault-tolerant gates with trivariate tricycle codes (2026). URL https://arxiv.org/abs/2508.08191. arXiv:2508.08191 [quant-ph].

[133] Fowler, A. G. & Devitt, S. J. A bridge to lower overhead quantum computation. *arXiv preprint arXiv:1209.0510* (2012).

[134] Fowler, A. G. Time-optimal quantum computation. *arXiv preprint arXiv:1210.4626* (2012).

[135] Xu, Q. *et al.* Fast and Parallelizable Logical Computation with Homological Product Codes. *arXiv preprint arXiv:2407.18490* (2024).

[136] Litinski, D. & Nickerson, N. Active volume: An architecture for efficient fault-tolerant quantum computers with limited non-local connections. *arXiv preprint arXiv:2211.15465* (2022).

[137] Litinski, D. A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery. *Quantum* **3**, 128 (2019).

[138] Symons, B. C., Rajput, A. & Browne, D. E. Sequences of bivariate bicycle codes from covering graphs. *arXiv preprint arXiv:2511.13560* (2025).

[139] Smarandache, R. & Vontobel, P. O. Quasi-cyclic LDPC codes: Influence of proto-and Tanner-graph structure on minimum Hamming distance upper bounds. *IEEE Trans. Inf. Th.* **58**, 585–607 (2012).

[140] Raveendran, N., Declercq, D. & Vasić, B. On the minimum distances of finite-length lifted product quantum ldpc codes. *arXiv preprint arXiv:2503.07567* (2025).

[141] Roffe, J., White, D. R., Burton, S. & Campbell, E. Decoding across the quantum low-density parity-check code landscape. *Phys. Rev. Res.* **2**, 043423 (2020).

[142] Ide, B., Gowda, M. G., Nadkarni, P. J. & Dauphinais, G. Fault-tolerant logical measurements via homological measurement. *arXiv preprint arXiv:2410.02753* (2024).

[143] Webster, P., Smith, S. C. & Cohen, L. Z. Explicit construction of low-overhead gadgets for gates on quantum ldpc codes. *arXiv preprint arXiv:2511.15989* (2025).

[144] Gu, B. *et al.* Qgpu: Parallel logic in quantum ldpc codes (2026). URL https://arxiv.org/abs/2603.05398. arXiv:2603.05398 [quant-ph].

[145] Pryadko, L. P., Shabashov, V. A. & Kozin, V. K. Qdistrnd: A gap package for computing the distance of quantum error-correcting codes. *arXiv preprint arXiv:2308.15140* (2023).

[146] Huang, S., Jochym-O'Connor, T. & Yoder, T. J. Homomorphic Logical Measurements. *arXiv:2211.03625* (2022).

[147] Sayginel, H., Koutsioumpas, S., Webster, M., Rajput, A. & Browne, D. E. Fault-tolerant logical clifford gates from code automorphisms. *PRX Quantum* **6**, 030343 (2025).

[148] Bravyi, S. & Kitaev, A. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Phys. Rev. A* **71**, 022316 (2005).

[149] Zhang, G., Zhu, Y., Yuan, X. & Li, Y. Constant-overhead magic state injection into qldpc codes with error independence guarantees. *arXiv preprint arXiv:2505.06981* (2025).

[150] Note that one could also teleport auto-corrected surface code $|\overline{T}\rangle$ states, which avoids the $\overline{S}$ corrections, at the price of using more surface code patches [110].

[151] Ismail, R. *et al.* Transversal star architecture for megaquop-scale quantum simulation with neutral atoms. *arXiv preprint arXiv:2509.18294* (2025).

[152] Vaknin, Y., Jacoby, S., Grimsmo, A. & Retzker, A. Efficient magic state cultivation on the surface code (2025). URL https://arxiv.org/abs/2502.01743. arXiv:2502.01743 [quant-ph].

[153] Gidney, C. Stim: a fast stabilizer circuit simulator. *Quantum* **5**, 497 (2021).

[154] Tremblay, M. A., Delfosse, N. & Beverland, M. E. Constant-Overhead Quantum Error Correction with Thin Planar Connectivity. *Physical Review Letters* **129**, 050504 (2022).

[155] Viszlai, J. *et al.* Matching generalized-bicycle codes to neutral atoms for low-overhead fault-tolerance. In *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 01, 688–699 (2025).

[156] Kang, M. *et al.* QUITS: A modular Qldpc code circUIT Simulator. *Quantum* **9**, 1931 (2025).

[157] Roffe, J. LDPC: Python tools for low density parity check codes (2022). URL https://pypi.org/project/ldpc/.

[158] Poulsen Nautrup, H., Friis, N. & Briegel, H. J. Fault-tolerant interface between quantum memories and quantum processors. *Nat. Comm.* **8**, 1321 (2017).

[159] Zhu, S., Sundaram, A. & Low, G. H. Unified architecture for quantum lookup tables. *Phys. Rev. Res.* **7**, 043230 (2025).

METHODS

Here we provide technical details on the codes, logic, and compilation schemes discussed in this work. In Appendix A, we describe and numerically benchmark the codes used in this work, including the lifted product code constructions and bivariate bicycle code constructions. In Appendix B, we describe how code surgery is performed, including a general description, the concrete constructions, and the associated resource costs and opportunities for future improvement. Appendix C describes the high-rate distillation procedure and the associated resource costs. Appendix D includes details of the numerical simulations. In Appendix E, we describe the compilation procedure for the space-efficient and balanced architectures, and their application to ripple-carry adders [53, 107] and table lookup [115] circuits. Finally, Appendix F describes the time-efficient architecture and the associated resource costs. Extended Data Table I summarizes the notation for different quantities used in this work.

Appendix A: Codes

Here we describe the high-rate quantum error-correcting codes used in this work and their functional roles. The codes are based on optimized constructions of lifted-product (LP) codes [44, 91] and prior constructions of bivariate bicycle (BB) codes [39, 138]. As summarized in Extended Data Table II, these codes offer different tradeoffs between block size, encoding rate, distance, and stabilizer weight. Here, the weight of a Pauli operator refers to the number of qubits for which it has nonzero support. These parameters are relevant to their practical performance, as the code distance bounds the logical error scaling with decreasing physical error rates [19], and the stabilizer weight is related to how many entangling gates are needed for stabilizer measurements.

1. Lifted-product codes

The LP code family [91] is obtained by taking the product of two classical codes with check matrices $A \in R^{r_A \times n_A}$ and $B \in R^{r_B \times n_B}$, where $R := \mathbb{F}_2[x]/(x^\ell + 1)$ is the univariate polynomial ring of order ℓ. Each entry of A is an element of R, which can be identified with a polynomial of degree at most $\ell - 1$ over $\mathbb{F}_2$, or equivalently, with an $\ell \times \ell$ circulant permutation matrix. In this work, we always choose $B = A^\dagger$, where $A^\dagger$ denotes the transpose of A with each polynomial entry $p(x)$ replaced by $p(x^{-1}) \bmod (x^\ell + 1)$. As the circulant permutation matrix corresponds to a cyclic shift in a line of atoms, this family of codes is readily implementable on reconfigurable atom arrays; see Ref. [44] for their detailed description and proposed implementation.

The resulting quantum code $\mathrm{LP}(A, A^\dagger)$ has parameters

$$[[n = (r_A^2 + n_A^2)\,\ell, \quad k \geq (n_A - r_A)^2\,\ell, \quad d]], \quad (A1)$$

with stabilizer weight $r_A + n_A$, where the distance satisfies the inequality [139, 140]

$$d \leq \min((r_A + 1)!, \quad (n_A + 1)!). \quad (A2)$$

As notation, we refer to an LP code built from an $r_A \times n_A$ seed matrix A over a ring of order ℓ and achieving quantum distance $\leq d$ as $\mathsf{lp}_d^{r_A, n_A}$.

In this work, we study codes with $r_A \times n_A = 3 \times 7$ and 3×5 seed matrices, corresponding to a distance of $d \leq 24$ from Eq. (A2). Because the true distance can be lower than this bound, we numerically obtain improved estimates of the distance upper bound. We use a decoder-based distance estimation [43] using the BP-OSD decoder [141] for both the distance of the Z-check matrix (d_Z) and the X-check matrix (d_X). Concretely, for d_Z (and similarly for d_X), let $L_X \in \mathbb{F}_2^{k \times n}$ be a basis for the logical X operators $\ker H_X / \mathrm{im}\, H_Z^\top$. For each trial, we sample $c \sim \{0, 1\}^k \setminus \{0\}$, set $e = c \cdot L_X \pmod 2$, and use BP-OSD to find a minimum-weight v satisfying $H_X v = 0$ and $e \cdot v = 1 \pmod 2$. Then $d_Z \leq \min_t |v_t|$. For this method, we consider at least $25,000$ trials for both d_Z and d_X. Below we describe the LP code instances studied in this work. The $\mathsf{lp}_{20}^{3,5}$ processor code and the memory codes $\mathsf{lp}_{16}^{3,7}$, $\mathsf{lp}_{20}^{3,7}$, $\mathsf{lp}_{24}^{3,7}$ are new to this work and are obtained using an LLM-assisted heuristic computer search [125].

$\mathsf{lp}_{20}^{3,5}$ **processor code.** The LP processor code for the balanced architecture is built from a 3×5 seed matrix A over the ring $R = \mathbb{F}_2[x]/(x^{33} + 1)$, so that $\ell = 33$. The seed matrix is

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & x^{14} & x^{19} & x^{11} & x^{26} \\ 1 & x^{13} & x^2 & x^{15} & x^{21} \end{pmatrix}, \quad (A3)$$

where we have written $x^0 = 1$. The LP construction in Eq. (A1) and numerical distance estimation yields a code with parameters

$$[[n = 1122, \quad k \geq 132, \quad d \leq 20]]. \quad (A4)$$

We find that the code has $k = 148$, exceeding the lower bound. The code has rate $k/n \approx 0.132$ and stabilizer weight $3 + 5 = 8$. The smaller block size and lower stabilizer weight make this code well-suited for use as a processor, where the overhead of performing logical gates scales with the code size (Appendix B). As shown in Extended Data Fig. 1a, $\mathsf{lp}_{20}^{3,5}$ achieves an extrapolated block error rate (per code cycle) of $\lesssim 10^{-11}$ at a physical error rate of $p = 0.1\%$.

$\mathsf{lp}_{16}^{3,7}$ **memory code.** The first memory code is built from a 3×7 seed matrix A over the ring $R = \mathbb{F}_2[x]/(x^{45} + 1)$,

Extended Data Table I. **Notation summary.** Summary of notation used in this work.

Notation	Description
$[[n_m, k_m, d_m]]$	Memory code parameters for the space-efficient and balanced architectures.
$[[n_p, k_p, d_p]]$	Processor code parameters for the space-efficient and balanced architectures.
$[[n_f, k_f, d_f]]$	Factory code parameters used in high-rate $8T$-to-CCZ distillation.
$[[n_s, k_s, d_s]]$	Surface code parameters used in magic state cultivation.
N_i	Size of a code indexed by i, including the data qubits and one basis (X or Z) of stabilizers, approximated by $N_i \simeq n_i + \lfloor (n_i - k_i)/2 \rfloor$.
N_m	Size of the memory code.
N_p	Size of the processor code.
N_f	Size of the factory codes.
N_s	Size of the cultivated surface codes.
$N_{\mathcal{A}}$	Size of the operation zone ancilla system.
$\mathcal{A}_m$	Ancilla system for the memory code.
$\mathcal{A}_p$	Ancilla system for the processor code.
$\mathcal{A}_f$	Ancilla system for the factory codes.
τ_s	Number of stabilizer measurement cycles during surgery PPMs, equal to $\frac{2}{3}d_p$.
$\tau_s^{\text{cult.}}$	Number of stabilizer measurement cycles to teleport cultivated states during high-rate distillation, equal to $\frac{2}{3}d_s$.
$\mathsf{lp}_{16}^{3,7}$	Small LP memory code with parameters $[[2610, 744, \leq 16]]$.
$\mathsf{lp}_{20}^{3,7}$	Medium LP memory code with parameters $[[4350, 1224, \leq 20]]$.
$\mathsf{lp}_{24}^{3,7}$	Large LP memory code with parameters $[[5278, 1480, \leq 24]]$.
$\mathsf{lp}_{20}^{3,5}$	LP processor code for the balanced architecture with parameters $[[1122, 148, \leq 20]]$.
bb_{18}	BB factory code and processor code for the space-efficient architecture with parameters $[[248, 10, \leq 18]]$.

so that $\ell = 45$. The seed matrix is

$$A = \begin{pmatrix} x^{29} & x^{21} & x^{31} & x^{15} & x^{37} & x^{25} & x^{27} \\ x^{13} & x^{25} & x^{19} & x^{26} & x^{11} & x^{18} & x^{29} \\ x^{31} & x^2 & x^{27} & x^{32} & x^{41} & x^{41} & x^{18} \end{pmatrix}. \qquad (A5)$$

The LP construction in Eq. (A1) and numerical distance estimation yields a code with parameters

$$[[n = 2610, \quad k \geq 720, \quad d \leq 16]]. \qquad (A6)$$

We find that the code achieves $k = 744$, slightly exceeding the lower bound guaranteed by the construction. The code has rate $k/n \approx 0.285$ and stabilizer weight $3 + 7 = 10$. As shown in Fig. 2a, $\mathsf{lp}_{16}^{3,7}$ achieves an extrapolated block error rate per cycle of $\approx 10^{-7}$ at $p = 0.1\%$.

$\mathsf{lp}_{20}^{3,7}$ **memory code.** The second memory code is built from a 3×7 seed matrix A over the ring $R = \mathbb{F}_2[x]/(x^{75} + 1)$, so $\ell = 75$. The seed matrix is

$$A = \begin{pmatrix} x^0 & x^{71} & x^{73} & x^{68} & x^{33} & x^{50} & x^{47} \\ x^{38} & x^{39} & x^{60} & x^{26} & x^{18} & x^1 & x^{23} \\ x^{73} & x^6 & x^5 & x^{42} & x^{20} & x^{22} & x^{73} \end{pmatrix}. \qquad (A7)$$

The LP construction in Eq. (A1) and numerical distance

estimation yields a code with parameters

$$[[n = 4350, \quad k \geq 1200, \quad d \leq 20]]. \qquad (A8)$$

We find that the code achieves $k = 1224$, slightly exceeding the lower bound guaranteed by the construction. The code has rate $k/n \approx 0.281$ and stabilizer weight $3 + 7 = 10$. As shown in Fig. 2a, $\mathsf{lp}_{20}^{3,7}$ achieves an extrapolated block error rate per cycle of $\lesssim 10^{-10}$ at $p = 0.1\%$.

$\mathsf{lp}_{24}^{3,7}$ **memory code.** The third memory code is built from a 3×7 seed matrix A over the ring $R = \mathbb{F}_2[x]/(x^{91} + 1)$, so $\ell = 91$. The seed matrix is

$$A = \begin{pmatrix} x^{57} & x^{75} & x^{42} & x^{80} & x^7 & x^{67} & x^{27} \\ x^{57} & x^{73} & x^{34} & x^{12} & x^{27} & x^{50} & x^{87} \\ x^{21} & x^{53} & x^{70} & x^{18} & x^1 & x^3 & x^{18} \end{pmatrix}. \qquad (A9)$$

The LP construction in Eq. (A1) and numerical distance estimation yields a code with parameters

$$[[n = 5278, \quad k \geq 1456, \quad d \leq 24]]. \qquad (A10)$$

We find that the code achieves $k = 1480$ logical qubits, again exceeding the lower bound. The code has rate $k/n \approx 0.280$ and stabilizer weight $3 + 7 = 10$. As shown

Extended Data Table II. **Code parameters.** Details of the codes used in this work, including the code parameters $[[n, k, d]]$, the stabilizer weight, and the encoding rate k/n.

Code	n	k	d	Weight	Rate
bb_{18}	248	10	≤ 18	6	0.04
$\mathsf{lp}_{20}^{3,5}$	1122	148	≤ 20	8	0.13
$\mathsf{lp}_{16}^{3,7}$	2610	744	≤ 16	10	0.29
$\mathsf{lp}_{20}^{3,7}$	4350	1224	≤ 20	10	0.29
$\mathsf{lp}_{24}^{3,7}$	5278	1480	≤ 24	10	0.28

in Fig. 2a, $\mathsf{lp}_{24}^{3,7}$ achieves an extrapolated block error rate per cycle of $\lesssim 10^{-11}$ at $p = 0.1\%$.

2. Bivariate bicycle codes

The BB code family [43] can be viewed as a special subfamily of LP codes. A BB code $\mathrm{LP}(a, b)$ is defined by two elements $a, b \in \mathbb{F}_2[x, y]/(x^l + 1, y^m + 1)$ in the bivariate polynomial ring with periods l and m. The resulting code acts on $n = 2lm$ physical qubits with stabilizer generators determined by the polynomials a and b. The BB code used in this work functions as both a processor and factory code.

bb_{18} **processor and factory code.** We use a BB code from Ref. [138] with parameters $[[248, 10, \leq 18]]$, defined by the polynomials

$$a = 1 + x^6 y + x^{27}, \quad b = y^2 + x^{15}y^3 + x^{24}, \quad \text{(A11)}$$

with $l = 31$ and $m = 4$. We refer to this code as bb_{18}. The code has rate $k/n = 10/248 \approx 0.04$ and stabilizer weight 6. As shown in Extended Data Fig. 1a, bb_{18} achieves an extrapolated block error rate (per code cycle) $\lesssim 10^{-11}$ at $p = 0.1\%$. Compared to the LP processor code $\mathsf{lp}_{20}^{3,5}$, this code has a significantly smaller block size and comparable distance, but encodes far fewer logical qubits per block. The choice between these two processor codes therefore introduces a tradeoff between per-block resource cost and the number of logical qubits per block.

Appendix B: Surgery

Universal computation on general qLDPC codes can be performed with logical Pauli product measurements (PPMs), which implement Clifford gates, and non-Clifford magic resource states. Code surgery [48, 49, 86, 142] is a method for implementing addressable PPMs on general qLDPC codes. First introduced in Ref. [48], it generalizes lattice surgery for topological codes [124] to general qLDPC codes. It has since been actively developed as a practical and flexible framework for fault-tolerant logical operations [49, 52, 86, 87, 112, 121, 122,

142].

Here we adapt these existing techniques to our architecture, analyze their concrete resource costs, and discuss their limitations and possibilities for further improvement. We refer readers to Sec. 3 of Ref. [87] for a comprehensive review of general code-surgery techniques, and provide a brief summary for completeness below.

1. Description

Given a qLDPC code $\mathcal{Q}$, a surgery gadget measures a collection of logical Pauli operators $\mathcal{L} = \{\bar{P} \in \bar{\mathcal{P}}_k\}$, where $\bar{\mathcal{P}}_k$ denotes the logical Pauli group acting on the k encoded qubits. This is achieved by coupling $\mathcal{Q}$ to an ancilla system $\mathcal{A}$ such that the eigenvalues of the logical operators in $\mathcal{L}$ are extracted non-destructively via measurements on $\mathcal{A}$ [48, 49, 86].
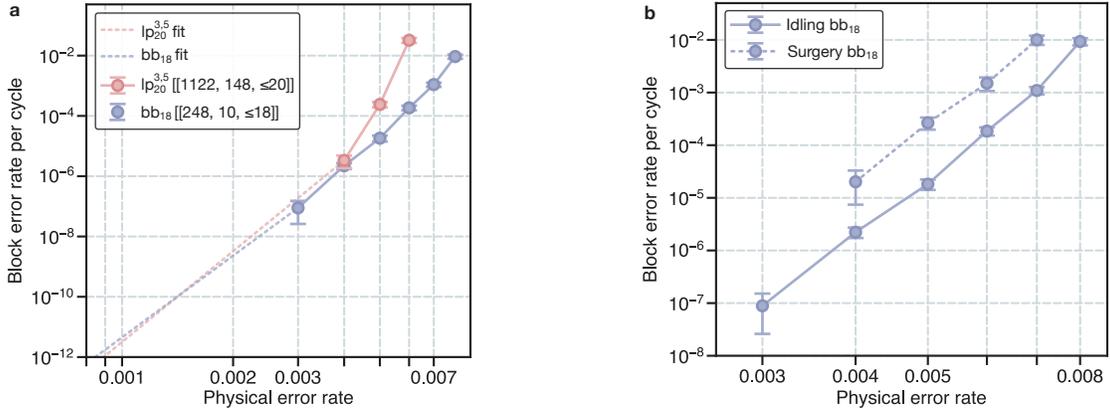
Here we briefly describe the procedure for measuring X-type logical operators. The procedures for other types of logical operators can be adapted analogously [87, 122]. For the data code $\mathcal{Q}(Q, S_X, S_Z)$, we denote by Q the set of qubits, and by S_X and S_Z the sets of X- and Z-checks, respectively. We construct an ancilla system $\mathcal{A}(Q', S'_X, S'_Z)$ with qubits Q' and check sets S'_X and S'_Z. The checks S_X and S_Z (resp. S'_X and S'_Z) act on Q (resp. Q') according to the parity-check matrices H_X and H_Z (resp. H'_X and H'_Z). We then perform the following surgery procedure to measure $\mathcal{L}$:

1. Initialize Q' in $|0\rangle$ states.

2. Measure τ_s cycles of checks $S_X \cup S'_X \cup S_Z \cup S'_Z$ that are now deformed to support on the *merged code* with qubits $Q \cup Q'$: in addition to the original connection, S'_X is also connected to Q and S_Z is also connected to Q' according to matrices f'_X and f_Z, respectively, depending on the target logicals to measure.

3. Detach $\mathcal{Q}'$ from $\mathcal{Q}$ by measuring Q' in the Z basis, followed by adaptive Pauli corrections on $\mathcal{Q}$.

The key step is measuring the checks of the *merged code* for τ_s cycles (step 2). We denote the merged code $\tilde{Q}(\tilde{Q}, \tilde{S}_X, \tilde{S}_Z)$ with check matrices

$$\tilde{H}_X = \begin{pmatrix} H_X & 0 \\ f'_X & H'_X \end{pmatrix}, \quad \tilde{H}_Z = \begin{pmatrix} H_Z & f_Z \\ 0 & H'_Z \end{pmatrix}. \quad \text{(B1)}$$

During this merging step, the information of $\mathcal{L}$ is extracted through measurements of the merged X-checks S'_X. Concretely, each $\bar{P} \in \mathcal{L}$ becomes an X-check in the row span of $\tilde{H}_X$. This requires $\langle \mathcal{L} \rangle = f'^T_X \ker(H'^T_X)$, meaning that the ancilla system is designed such that the kernel of H'^T_X maps exactly to the target logical operators of the data code. We focus on protocols in which f'^T_X has the structure that a subset of qubits in Q is connected one-to-one with a subset of X-checks in S'_X. In this case,

Extended Data Fig. 1. **Processor code and surgery performance. a,** Block error rate per cycle as a function of physical error rate for the $\mathsf{lp}_{20}^{3,5}$ (pink) and bb_{18} (blue) codes. A least-square fit to both the bb_{18} and $\mathsf{lp}_{20}^{3,5}$ data yields a slope greater than $d/2$, the theoretical maximum value in the limit that the physical error rate $p \to 0$. We therefore fit the data point at the smallest physical error rate of $\mathsf{lp}_{20}^{3,5}$ (pink dashed line) the form $y = ax^{10}$, where we find $a = 18.5$. Using the same procedure for bb_{18} (blue dashed line) to fit a line of the form $y = ax^9$, we find $a = 15.65$. **b,** Block error rates per cycle versus physical error rate for the bb_{18} code, either idling for 9 cycles (solid line) or performing a surgery measurement of a high-weight logical operator (Table III) with $\tau_s = 15$ cycles (dashed line). Surgery failure rates are averaged over X- and Z-basis initialization and measurement experiments.

the logical operators to be measured are fully specified by the kernel of the classical code $H_X'^T$. Depending on the dimension of $\ker(H_X'^T)$, we classify surgery gadgets into either *low-rate surgery* or *high-rate surgery*, described below. The fault-tolerance of the surgery gadget is ensured if the following conditions hold [49, 52, 87]: (i) the merged code $\tilde{\mathcal{Q}}$ has distance $\tilde{d} = \Theta(d)$, (ii) the merged code remains qLDPC, and (iii) $\tau_s = \Theta(d)$. We discuss how these criteria can be satisfied below. Note that the merged code can be a subsystem code with additional gauge qubits arising from the ancilla system $\mathcal{A}$ [48, 52, 142], and in general we are referring to the distance of the subsystem code when we refer to the distance of the merged code $\tilde{d}$.

Low-rate surgery: $\dim \ker H_X'^T = 1$. In this case, we always measure one logical operator $\bar{P}$ per gadget. The representative constructions [49, 86, 87] use a connected graph $G(V, E)$, with a set of vertices V representing S_X' and edges E representing Q', for the ancilla system. This naturally guarantees $\dim \ker H_X'^T = 1$ since $\ker H_X'^T$ is simply the number of connected components of G. S_Z' is chosen to be a cycle basis of G. The distance of the merged code is guaranteed to be $\tilde{d} \geq d$ if the boundary Cheeger constant of the graph G is ≥ 1 [112]; the merged code can also be made qLDPC using graph-theoretical techniques such as cellulation and decongestion [49]. These conditions can be satisfied using an ancillary graph of size $|\mathcal{A}| = \mathcal{O}(|P| \log^3 |P|)$, where $|P|$ denotes the physical Hamming weight of $\bar{P}$. Such a graph-based gadget can also be used to measure $\bar{P}$ of any type, i.e. a product of single-qubit logical Pauli $\bar{X}$, $\bar{Y}$, and $\bar{Z}$ operators. A typical approach [49, 86], which we adopt in this work, constructs the ancilla system $\mathcal{A}$ *dynamically* depending on the target logical operator $\bar{P}$ to be measured. More modular alternatives that employ a fixed

ancilla system for measuring different logical operators have also been developed. For example, the extractor construction [87] enables such a modular design at the cost of a larger ancilla overhead of $O(n \log^3 n)$. Another approach leverages codes with rich automorphism groups [49, 143], which reduces the task to measuring lower-weight "seed" logical operators, potentially leading to lower overhead.

High-rate surgery: $\dim \ker H_X'^T = t > 1$. This gadget measures up to t logical operators in parallel. Note that this entails that the ancilla system $\mathcal{A}$ is either a graph with multiple connected components (i.e. a union of disjoint, graph-based ancilla systems) or a hypergraph. Ref. [122], improving upon Ref. [121], introduces a general scheme that can measure an arbitrary set of logical operators (including $\bar{Y}$-operators and a mixture of $\bar{X}$-type and $\bar{Z}$-type operators) of an $[[n, k, d]]$ qLDPC code with logically disjoint support with a provable bound on the ancilla size $\tilde{\mathcal{O}}(k\omega)$, where ω is the maximum physical weight of any logical Pauli being measured; Refs. [52, 79, 127, 144] present techniques for measuring certain restricted patterns of $\bar{Z}$-type and $\bar{X}$-type logicals with an ancilla size $\mathcal{O}(n)$, i.e. bounded by the code block size, for $t = O(k)$. More generally, Ref. [52] also presents a randomized algorithm that can measure a more flexible set of $\bar{Z}$-type or $\bar{X}$-type logicals with low overhead and numerically observes that it can measure up to $t \sim k$ logicals for a variety of codes with an ancilla size comparable to the original code size. Here, with a slight abuse of definitions, we refer to any scheme that can measure multiple logicals in parallel using an ancilla whose size is bounded by that of the data code, i.e. $|\mathcal{A}| = \tilde{O}(n)$, as a *high-rate surgery* gadget. Note that, although we do not know such a construction yet that can measure an arbi-

trary set of logically non-overlapping logicals (including Y- and XZ-type) for any code with a guaranteed low overhead, we expect that more advanced schemes based on Refs. [52, 113, 127, 144] can approach this goal, at least for certain structured codes.

So far, we have focused on surgery gadgets for a single code block. We now briefly discuss how to perform logical PPMs across multiple code blocks. In principle, one may treat several code blocks as a single combined code by taking their union and then apply the standard techniques described above. Alternatively, to measure a Pauli operator of the form $\bar{P}\bar{P}'$, where $\bar{P}$ and $\bar{P}'$ are supported on different code blocks (possibly of different code families), one can construct two independent ancilla systems $\mathcal{A}$ and $\mathcal{A}'$ for measuring $\bar{P}$ and $\bar{P}'$, respectively, and then connect them through a bridge system $\mathcal{B}$ [49, 112] to measure their product. Using the construction algorithm of Ref. [112], the bridge system requires as few as d qubits and $d-1$ checks, where d is the minimum distance of the two codes involved. When either $\mathcal{A}$ or $\mathcal{A}'$ is much larger than d, the additional qubit overhead of the bridge system is negligible compared to the size of the original ancilla systems.

2. Concrete construction and benchmarking

Using the general procedure described in the previous section, we present concrete constructions of surgery gadgets for the space-efficient and balanced architecture in this section. Recall that these architectures consist of a $[[n_m, k_m, d_m]]$ memory code, a $[[n_p, k_p, d_p]]$ processor code, and several $[[n_f, k_f, d_f]]$ factory codes (see Fig. 2b and Extended Data Table I). As discussed in Appendix E, the logical PPMs required for computation have the following forms: $\bar{Z}_i\bar{P}$ or $\bar{P}$, where $\bar{Z}_i$ is a single-qubit logical Pauli $\bar{Z}$ operator acting on either the memory or a factory code, and $\bar{P}$ is an arbitrary k_p-qubit logical Pauli operator on the processor code; or $\bar{X}_j$, a single-qubit logical Pauli $\bar{X}$ operator acting on either the memory or a factory code.

Accordingly, we introduce three ancilla systems $\mathcal{A}_m$, $\mathcal{A}_p$, and $\mathcal{A}_f$ for measuring logical Pauli operators on the memory, processor, and factory codes, respectively. In addition, two bridge systems are used to connect $\mathcal{A}_m$ and $\mathcal{A}_p$, and $\mathcal{A}_p$ and $\mathcal{A}_f$, respectively. These PPMs are implemented using *low-rate surgery* techniques (see the previous section) such that one logical operator of the above form is measured at a time using a single surgery gadget.

Using standard surgery constructions [49, 52, 86], which dynamically generate ancilla systems depending on the target logical operator, the number of distinct ancilla systems scales as $O(k_m)$, $O(k_f)$, and $O(\exp(k_p))$ in the worst case for $\mathcal{A}_m$, $\mathcal{A}_f$, and $\mathcal{A}_p$, respectively. In architectures with fixed connectivity, such as superconducting circuits [46], each distinct ancilla system would need to be physically constructed and con-

nected to the data codes, significantly increasing the required hardware footprint. In contrast, by leveraging the reconfigurable connectivity of atom arrays, it suffices to reserve an operation zone with enough qubits to accommodate the largest ancilla configuration, namely $\max_{\mathcal{A}_m, \mathcal{A}_p, \mathcal{A}_f}(|\mathcal{A}_m| + |\mathcal{A}_p| + |\mathcal{A}_f|)$ (note that here we do not include ancilla qubits for the bridge systems, which are subleading in size). The ancilla systems can then be dynamically reconfigured within this region as needed. We note that this space saving comes at the cost of increased overhead in classical processing and control, as one must construct the distinct surgery gadgets in real-time and adaptively program the corresponding ancilla configurations. We discuss possible approaches to reducing these classical overheads and further improving resource efficiency in Appendix 3.

As summarized in Extended Data Table III, we construct and benchmark representative instances of the ancilla systems described above. The first and last rows present $\mathcal{A}_m$ and $\mathcal{A}_f$, used to measure a single-qubit logical X operator on the $\mathsf{lp}_{20}^{3,7}$ and $\mathsf{lp}_{24}^{3,7}$ memory codes and the bb_{18} factory code, respectively. Minimizing the size of these ancilla systems requires selecting a logical basis composed of conjugate pairs of single-qubit logical operators with low physical weight. For the bb_{18} code, we numerically identify such a basis in which nine logical qubits have weight-18 logical operators, while one has a weight-20 logical operator. Finding a comparable basis for the larger memory LP codes is more challenging numerically. Upcoming work [113] provides an algebraic construction of a low-weight basis for LP codes, in which the 1,220 (resp. 1,476) logical qubits in $\mathsf{lp}_{20}^{3,7}$ (resp. $\mathsf{lp}_{24}^{3,7}$) have physical weight at most 200 (resp. 224). Furthermore, by exploiting code automorphisms, measuring any of the above low-weight single-qubit logical $\bar{X}$ operators of $\mathsf{lp}_{20}^{3,7}$ (resp. $\mathsf{lp}_{24}^{3,7}$) requires only 112 (resp. 144) distinct surgery gadgets [113]. We report, in the first and last rows of Extended Data Table III, the gadgets that measure the maximum-physical-weight single-qubit logical $\bar{X}$ operators for $\mathsf{lp}_{20}^{3,7}$, $\mathsf{lp}_{24}^{3,7}$ and bb_{18}, respectively. The middle rows present instances of $\mathcal{A}_p$ used to measure high-weight logical $\bar{X}$ operators on the bb_{18} and $\mathsf{lp}_{20}^{3,5}$ processor codes. Each operator is selected as the maximum-physical-weight example among 10^5 randomly sampled logical multi-qubit $\bar{X}$ operators, with logical (resp. physical) weights 11 (resp. 104) for bb_{18}, and 69 (resp. 460) for $\mathsf{lp}_{20}^{3,5}$.

All ancilla systems are constructed using standard graph-based surgery techniques [49, 86], which will be further detailed in Ref. [113]. The distances of the surgery gadgets are estimated using the QDistRnd package [145] with at least 100,000 trials. Extended Data Figure 1b shows the logical error rates of the surgery gadget listed in Extended Data Table III that measures a high-weight logical operator on the bb_{18} code, corresponding to $\mathcal{A}_p$ in the space-efficient architecture. With $\tau_s = 15 \approx 2d/3$, the surgery error rates are within an order of magnitude of the bb_{18} block failure rate at idling,

demonstrating fault tolerance of the surgery construction. We expect similar behavior for the larger gadgets in Extended Data Table III, and leave detailed simulations to future work.

Although Extended Data Table III benchmarks ancilla systems for measuring $\bar{X}$-type logical operators, the same constructions readily extend to other logical types (e.g., $\bar{Y}$) by modifying how the ancilla systems couple to the data codes [87]. Accordingly, we estimate the space cost of the ancilla systems in our architecture (i.e., the size of the operation zone), $\max_{\mathcal{A}_m, \mathcal{A}_p, \mathcal{A}_f} \left(|\mathcal{A}_m| + |\mathcal{A}_p| + |\mathcal{A}_f| \right)$, based on the examples in Extended Data Table III.
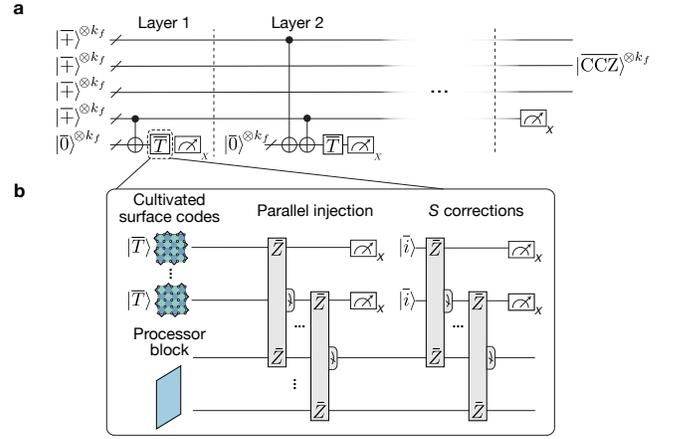
3. Resource costs and future improvements

We now summarize the time and space costs of the surgery instructions used in this architecture. Based on the examples in Extended Data Table III as well as the analysis in the previous sections, we estimate that the ancillary systems required to perform all relevant surgery operations in our architecture occupy $N_\mathcal{A} = 894$ and $N_\mathcal{A} = 1{,}874$ (resp. $N_\mathcal{A} = 924$ and $N_\mathcal{A} = 1{,}904$) qubits for the space-efficient and balanced architectures, respectively, when using the $\mathsf{lp}_{20}^{3,7}$ (resp. $\mathsf{lp}_{24}^{3,7}$) memory code (see also Extended Data Table IV). We assume that each surgery gadget is implemented in $\tau_s \approx 2d/3$ code cycles, where d denotes the distance of the processor code. This choice balances the space-like and time-like logical error rates of the gadget, thereby approximately minimizing the overall logical error rate. See Sec. 2 for numerical justification of this choice.

There are several avenues to reduce the classical overhead associated with constructing and implementing surgery gadgets, as well as further lowering the overall resource costs. First, the number of distinct surgery gadgets could be reduced using extractor-style constructions [87], or more modular approaches that rely on a small set of fixed gadgets and bridge systems by exploiting symmetries of the data code, particularly automorphisms [37, 49, 143]. For codes with sufficiently rich symmetries, the latter approach may enable a fixed ancilla system that is even smaller than the data block while still supporting measurements of arbitrary logical operators [143]. In addition, alternative techniques beyond code surgery may further reduce overhead, including homomorphic measurements [47, 51, 146] and in-block transversal gates [47, 88, 147]. Some of these directions will be explored in upcoming work [113] for the lifted-product code family.

Appendix C: Magic

Universal quantum computation requires non-Clifford gates in addition to Clifford gates; typically, $\overline{T}$ or $\overline{CCZ}$ gates are used. Non-Clifford gates can be executed by generating magic resource states (e.g., $|\overline{T}\rangle = \overline{T}\,|\overline{+}\rangle$ and

Extended Data Fig. 2. **Parallel magic state distillation.** We distill $k_f\ |\overline{CCZ}\rangle$ states using five copies of $[[n_f, k_f, d_f]]$ factory codes. Low-error cultivated surface-code $|\overline{T}\rangle$ states are loaded into one processor code using parallel surgery [149]. Transversal CNOT gates between this code and the four remaining processor codes are used to perform the PPMs in $8T$-to-CCZ distillation [110], outputting three blocks of high-fidelity $|\overline{CCZ}\rangle$ states.

$|\overline{CCZ}\rangle = \overline{CCZ}\,|\overline{+}\rangle^{\otimes 3}$) and performing gate teleportation with conditional Clifford feedforward gates [148]. In this work, we implement Toffoli gates by distilling high-fidelity $|\overline{CCZ}\rangle$ states and teleporting them into the computation using logical PPMs [110, 136].

Historically, generating magic resource states has been among the most resource-demanding subroutines for implementing large-scale logical algorithms for planar surface-code architectures [34, 110]. Recent techniques such as magic state cultivation [50] substantially reduce the cost of generating magic states in planar architectures. However, cultivation uses surface codes with low encoding rates, and cultivation alone cannot always achieve the logical error rates required for large-scale algorithms [18]. Here, we describe a protocol that generates many magic states in parallel with low overhead and logical error rates by combining surface-code cultivation with high-rate codes.

1. Description

As illustrated in Extended Data Fig. 2, the high-rate magic state distillation procedure distills cultivated $|\overline{T}\rangle$ states on small surface codes into high-fidelity $|\overline{CCZ}\rangle$ states hosted in three high-rate factory code blocks with parameters $[[n_f, k_f, d_f]]$, using the $8T$-to-CCZ distillation circuit in Ref. [110]. The distillation circuit has eight layers of gates, each with a similar structure. For a given layer, we use an ancillary factory code block, denoted as the $\overline{T}$ block, to implement parallel logical Pauli product rotations on the remaining four processor code blocks (see Fig. 16 of Ref. [110] for the concrete circuit). This is done

Extended Data Table III. **Surgery systems.** Ancilla systems for measuring the relevant PPMs on different zones and codes. For each ancilla system, we report the target logical operators it measures and their logical weights, the number of ancillary qubits, X stabilizers, and Z stabilizers, as well as the resulting degree and the distance upper bound of the merged code.

Zone	Codes	PPMs	(Qubits, X-checks, Z-checks)	Degree	Distance
Memory	$[[4350, 1224, \leq 20]]$-$\mathsf{lp}_{20}^{3,7}$	$\bar{P}, |\bar{P}| = 1$	(342, 200, 143)	12	≤ 20
	$[[5278, 1480, \leq 24]]$-$\mathsf{lp}_{24}^{3,7}$	$\bar{P}, |\bar{P}| = 1$	(364, 208, 157)	12	≤ 22
Processor	$[[248, 10, \leq 18]]$-bb_{18}	$\bar{P}, |\bar{P}| = 9$	(189, 104, 86)	9	≤ 18
	$[[1122, 148, \leq 20]]$-$\mathsf{lp}_{20}^{3,5}$	$\bar{P}, |\bar{P}| = 69$	(813, 460, 357)	10	≤ 20
Resource	$[[248, 10, \leq 18]]$-bb_{18}	$\bar{P}, |\bar{P}| = 1$	(39, 20, 20)	7	≤ 18

using up to four transversal CNOT gates between the $\overline{T}$ block and the remaining processor code blocks, followed by a transversal faulty $\overline{T}$ measurement [110].

As shown in Extended Data Figure 2b, the transversal faulty $\overline{T}$ measurements on the $\overline{T}$ block are implemented by parallel teleportation of k_f cultivated $|\overline{T}\rangle$ states, each hosted in a small $[[n_s, k_s, d_s]] = [[d_s^2, 1, d_s]]$ surface-code patch. For the parallel teleportation we use parallel code surgeries [149] (Appendix B), followed by local $\overline{S}$ corrections via teleportation of $|\overline{Y}\rangle$ states. The $|\overline{Y}\rangle$ states can be prepared using a subset of the patches used for cultivation, for example using fold-transversal gates in depth one [150]. Finally, we perform transversal X-basis measurements on the $\overline{T}$ block, followed by Pauli feedback on the other four factory blocks. This procedure is repeated eight times, once for each layer of logical Pauli-product rotations in Fig. 16 of Ref. [110]. The resulting circuit generates k_f $|\overline{\text{CCZ}}\rangle$ states uniformly distributed among three factory code blocks, i.e. $|\overline{\text{CCZ}}\rangle^{\otimes k_f}$. The correctness of the generated magic states is heralded by the fourth factory block being measured successfully in $|\overline{+}\rangle^{\otimes k_f}$ in a transversal X-basis measurement.

Using factory codes with a sufficiently large distance $d_f \gg d_s$, noise on the Clifford operations such as transversal CNOT gates can be neglected to good approximation [110], and the main faulty operations in the circuit are the $\overline{T}$ gates on the $\overline{T}$ block via teleporting the surface code $|\overline{T}\rangle$ states. As such, the logical error rates of the output $|\overline{\text{CCZ}}\rangle$ states depend primarily on the error rates of these noisy $\overline{T}$ gates, which depend on the fidelity of the cultivated $|\overline{T}\rangle$ states. In addition, there are also extra errors introduced to the $\overline{T}$ factory code block when performing the parallel surgeries between the high-distance factory code and the low-distance surface codes. For example, in order to avoid introducing logically correlated errors on the factory code due to low weight physical faults during the surgery, the protocol in Ref. [149] requires a surgery ancilla system with size $\tilde{O}(k_f d_f)$ and $O(d_f)$ code cycles, which exponentially suppresses the correlated error by d_f, where $\tilde{O}(\cdot)$ indicates scaling up to logarithmic factors. However, because we are performing transversal distillation across different high-rate blocks rather than within a high-rate code block, we can allow for correlated errors within the $\overline{T}$ block. This is because

distillation across blocks effectively involves k_f parallel distillation factories.

As such, we are only concerned about the *marginal error rate* of each $\overline{T}$ gate in a $\overline{T}$ factory code block. For this, it is sufficient to use surgery ancillas of size $\tilde{O}(k_f d_s)$ and $\tau_s^{\text{cult.}} = O(d_s)$ code cycles, which exponentially suppresses the extra (possibly correlated) errors during the surgery process by d_s. We assume that each layer of surgery takes $\tau_s^{\text{cult.}} = 2d_s/3$ code cycles, which often minimizes the total logical error rates [49], and the final marginal error rate for the $\overline{T}$ gate is approximately $2p_T$, where p_T is the error rate of the cultivated surface code $|\overline{T}\rangle$ states. We then estimate that the logical error rate of each output $|\overline{\text{CCZ}}\rangle$ state is $p_{\text{CCZ}}^L \approx 28(2p_T)^2$ [110]. The success probability of the protocol scales linearly with p_T. Since we typically have $p_T \ll 1$, the success probability that the entire block succeeds is very close to 1.

The total time cost is approximately $8 \times (\tau_T + 2\tau_s^{\text{cult.}})$, where τ_T denotes the time to cultivate each surface code $|\overline{T}\rangle$ state. As for space cost, the surgery ancilla is typically much smaller than the ancilla required for performing fully fault-tolerant logic for other modules of our architecture, e.g. the processor code or the memory code, so we neglect it here and do not include it into the space cost of the operation zone. This can also be justified if we can pipeline the surgery operations across the full architecture such that a subset of the ancillary qubits in the operation zone can be reconfigured for executing the surgeries in the factory. As such, and based on the analysis above, we estimate that the space cost of the entire factory is $5N_f + k_f N_s$.

2. Resource costs

Here we compute the concrete resource costs for the magic factory described in the main text for the space efficient and balanced architectures. For the factory code, we select the bb_{18} code (Extended Data Table II) with $N_f = 367$. We choose the distance of the surface code to be $d_s = 7$ so that it has logical error rate $\lesssim 10^{-6}$ [151]. We consider the scheme in Ref. [111] for cultivating $|\overline{T}\rangle$ states with $p_T \approx 10^{-6}$ at physical error rates of 0.1%. This approach involves injecting a noisy $|\overline{T}\rangle$ state with fault distance 1 into a small $d = 3$ rotated surface code,

then measuring the fold-transversal Hadamard operator twice (each measurement followed by a stabilizer measurement cycle). Finally, the code is grown from $d = 3$ to $d = 5$ using a unitary growth procedure, then to $d = 7$ using stabilizer measurements. Postselection is applied between the measurement and growth stages, then after the growth stage. For physically motivated noise models (see Fig. 3 of Ref. [111]), in expectation fewer than two total attempts are required for the cultivated state to pass all postselection tests to reach error rates of $\lesssim 10^{-6}$. Note that our choice of final code distance $d_s = 7$ is likely conservative because $\gtrsim 50\%$ of errors are atomic loss in realistic settings, and such heralded errors both improve the threshold by a factor of $\approx 2\times$ and enable better postselection [152].

Because the injection and cultivation stages occur at a smaller code size, extra surface code blocks can be prepared in parallel such that the success probability of this stage is close to one, without exceeding the maximum space cost after growth. The total depth, from injection to growth, is approximately four cycles: one cycle for injection, one for cultivation, and two for growth. The full process requires fewer than two attempts, ≈ 1.25, in expectation. As such, we estimate that cultivating each $|\overline{T}\rangle$ state takes $\tau_T \approx 5$ code cycles, and each surgery layer takes $\tau_s^{\text{cult.}} \approx 2d_s/3 \approx 5$ stabilizer measurement cycles. With these choices, our factory produces $k_f = 10$ $|\overline{\text{CCZ}}\rangle$ states, each with an error rate $p_{\text{CCZ}}^L \approx 10^{-10}$, using 2,565 total qubits in $8\times(5+2\cdot5) \approx 120 = 6d_p$ cycles, for $d_p = 20$ (see also Extended Data Table IV). Note that the time cost per $|\overline{\text{CCZ}}\rangle$ state is $120/k_f = 12 < d_p$. As such, in our space-efficient or balanced architecture, where we consume these $|\overline{\text{CCZ}}\rangle$ states sequentially, we will neglect the time cost for producing each $|\overline{\text{CCZ}}\rangle$ state.

Appendix D: Numerical simulations

We numerically assess the performance of our fault-tolerant architecture under circuit-level noise using Stim [153]. We focus on simulating the performance of the codes used in our architecture under repeated stabilizer measurements, as well as the surgery gadgets presented in Appendix B.

Noise model. We use a circuit-level depolarizing noise model parameterized by a single physical error rate p, consistent with prior work [44, 80, 89]. The noise channels are applied as follows.

- *Two-qubit gate noise.* Each CNOT gate is followed by a two-qubit depolarizing channel with error rate p, which applies each of the 15 nontrivial two-qubit Pauli operators with equal probability $p/15$.

- *State preparation noise.* Each state preparation ($|0\rangle$ or $|+\rangle$) is followed by a single-qubit depolarizing channel with error rate p.

- *Measurement noise.* Each measurement is preceded

by a single-qubit depolarizing channel with error rate p.

Syndrome extraction circuit. For each code, we construct a syndrome extraction circuit that measures the X-type and Z-type stabilizers separately. For each stabilizer type, we determine the gate scheduling, specifically the ordering of CNOT gates at each time step, using the coloration scheduling method [154]. This approach performs an edge coloring of the bipartite Tanner graph associated with the check matrix, ensuring that no data qubit participates in more than one gate per time step. As a result, the circuit requires $\Delta_X + \Delta_Z$ layers of CNOT gates for a code whose X and Z-check matrices have degrees Δ_X and Δ_Z, respectively. We use this generic scheduling throughout both the memory experiments and the surgery experiments described below whenever stabilizers of a (possibly deformed) quantum code need to be measured. We adopt this generic coloration scheduling primarily for ease of simulation. In practice, more structured approaches could be used, such as the "product-coloration" scheduling for LP codes [44], translationally invariant scheduling for BB codes [43, 155] that is directly compatible with current neutral-atom devices, or shorter-depth schedules that interleave the measurements of X and Z-checks [44, 156].

Here we give a detailed step-by-step of the implementation of the circuit [44, 153, 154]. First, the X-type ancillas are initialized in the $|+\rangle$ state via a Hadamard gate, followed by preparation noise. A sequence of CNOT gates is then applied between each X-ancilla (as control) and its supported data qubits (as targets), with the gate schedule determined by the edge coloring. Next, the Z-type ancillas are initialized in the $|0\rangle$ state, followed by preparation noise, and CNOT gates are applied with each data qubit as control and the corresponding Z-ancilla as target. Finally, all ancillas are measured and reset.

Memory experiment. We simulate a quantum memory experiment consisting of $d/2$ stabilizer measurement cycles, where d is the code distance. The logical qubits are initialized in the $|+\rangle^{\otimes k}$ state via transversal X-basis preparation of all data qubits. In the first cycle, detectors are defined by the raw X-syndrome outcomes. In subsequent cycles, detectors are defined by the difference between consecutive X-syndrome outcomes, which flags any change due to errors occurring between cycles. After the final cycle, all data qubits are measured in the X basis with measurement noise applied. The final-cycle detectors compare the data qubit measurement outcomes against the last cycle by reconstructing the X-stabilizer values from the data. Logical observables are defined by the logical $\bar{X}$ operators of the code.

Surgery experiment. We simulate a surgery gadget that measures a set of t logical $\bar{X}$ operators $\mathcal{L} = \{\bar{P}_i\}_{i=1}^t$, following the general three-step protocol described in Sec. B. Recall that a surgery gadget consists of a k-logical-qubit data code with physical qubits Q together with an ancilla system with physical qubits Q'. During the protocol, the stabilizer checks of the merged code are

measured for τ_s cycles. For each gadget we perform two experiments. In both experiments the ancilla qubits Q' are initialized and measured in the Z basis, while the data qubits Q are initialized and measured in either the X or the Z basis.

In the X-basis experiment, we define $k + t$ logical observables: the final logical $\bar{X}$ operators of the data code, obtained from the parities of the final single-qubit X measurements on Q, together with the outcomes of the target logical operators $\mathcal{L}$, extracted from the parities of the merged-code X-checks in the first stabilizer measurement cycle. Failures of these observables characterize space-like logical $\bar{Z}$ errors (determined by the Z distance of the merged code) as well as time-like logical errors corresponding to incorrect measurement of the target logical operators. In the Z-basis experiment, we define $k - t$ logical observables corresponding to the logical $\bar{Z}$ operators of the data code that commute with $\mathcal{L}$. Failures of these observables characterize space-like logical $\bar{X}$ errors, determined by the X distance of the merged code.

Decoding. Decoding is performed using an ensemble of belief propagation with localized statistics decoders (BP-LSD) [45], implemented via the `ldpc` package [157]. Each decoder instance uses min-sum belief propagation with serial scheduling for 100 iterations, using the default annealing schedule for the min-sum scaling factor provided by the `ldpc` package (`ms_scaling_factor = 0.0`). The LSD is performed using an exhaustive search of order 5 (`lsd_method = lsd_e, lsd_order = 5`). To improve decoding performance, we run an ensemble of five decoder instances per syndrome, each operating under a different channel probability model: (i) the nominal error probabilities $\vec{p}$, (ii) a randomized serial schedule with the nominal probabilities, (iii) optimistic probabilities $0.8\,\vec{p}$, (iv) pessimistic probabilities $1.2\,\vec{p}$, and (v) thermally perturbed probabilities $\vec{p} \circ (1 + \vec{\eta})$, where $\eta_i \sim \mathcal{N}(0, 0.04)$ independently for each error mechanism. For each syndrome, all five decoders produce a candidate error vector. Candidates that satisfy the syndrome are retained, and the one with the lowest log-likelihood ratio weighted cost

$$C(\vec{e}) = \sum_i \left| \log \frac{1 - p_i}{p_i} \right| \cdot e_i \qquad \text{(D1)}$$

is selected, where p_i is the prior error probability of the i-th error mechanism and $e_i \in \{0, 1\}$ indicates whether that mechanism is included in the correction. If no candidate satisfies the syndrome, a logical failure is recorded. The decoder is designed by an LLM-assisted heuristic computer search [125].

Logical error rate. We report the block logical error rate: a failure is recorded whenever the selected correction, composed with the true error, acts nontrivially on any logical observable.

Appendix E: Space-efficient and balanced architectures

In this section, we present a general compilation scheme for implementing any Clifford + Toffoli circuit on our space-efficient and balanced architectures based on the logical instructions in Appendices B and C. We then apply this general compilation strategy to two algorithmic subroutines—ripple-carry adders [53, 107, 114] and unary lookup tables [115]—which are the key elementary building blocks of the cryptographic applications considered in this work, and estimate their time costs. Finally, we estimate the runtimes for RSA–2048 and ECC–256 using our space-efficient and balanced architectures.

Our space-efficient or balanced architectures consist of:

- A $[[n_m, k_m, d_m]]$ memory code, where k_m is larger than the logical footprint of the entire algorithm.

- A $[[n_p, k_p, d_p]]$ processor code.

- Three $[[n_f, k_f, d_f]]$ factory codes hosting k_f copies of logical $|\overline{\text{CCZ}}\rangle$ states distilled using ancillary factory code blocks, where each $|\overline{\text{CCZ}}\rangle$ state is encoded across codes.

- Additional ancillary codes for high-rate distillation, including two $[[n_f, k_f, d_f]]$ factory codes and k_f $[[n_s, 1, d_s]]$ surface codes for cultivation.

Given a general Clifford + Toffoli circuit on k_m qubits, we serialize it into sub-circuits $\{C_i\}$, where each C_i is a m_i-qubit circuit containing β_i Toffoli gates, γ_i mid-circuit Pauli measurements, and arbitrary Clifford gates. We require that each sub-circuit can fit inside the processor code block, i.e. $m_i \leq k_p$ for all C_i. We use $\bar{P}$, $\bar{P}'$, and $\bar{P}''$ to indicate a logical Pauli operator of the memory code, the processor code, and the factory code, respectively. Then, as illustrated in Extended Data Figure 3, we implement each C_i with the following steps:

1. Teleport the m_i logical qubits in the memory code to the processor code by sequentially measuring $\{\bar{Z}_{I_j}\bar{Z}'_j\}_{j \in [m_i]}$ and then $\{\bar{X}_{I_j}\}_{j \in [m_i]}$. Here, I_j denotes the index of the j-th qubit among the m_i qubits in the memory code that C_i is supported on.

2. Perform a m_i-qubit Pauli-based computation on the processor code, where Cliffords are pushed to the end and each Toffoli gate is implemented by three $\bar{P}'\bar{Z}''$ measurements between the processor code and one of the factory codes, where $\bar{P}'$ is a m_i-qubit Pauli operator. In addition, each mid-circuit Pauli measurement will be transformed to a high-weight PPM and also executed sequentially.

3. Teleport the m_i qubits back into the memory code by sequentially performing $\bar{Z}_{I_j}\bar{P}'_j$ between the processor and the memory code, followed by m_i m_i-qubit measurements on the processor code.

The above protocol extensively utilizes two key subroutines:

1. *Measurement-based teleportation:* logical qubit i is teleported to logical qubit j (potentially across different codes) by preparing $|\overline{+}\rangle_j$, performing a $\bar{Z}_i\bar{Z}_j$ PPM followed by a $\bar{X}_i$ PPM, and applying the corresponding logical Pauli corrections (see Fig. 3 of Ref. [158]).

2. $|\overline{\text{CCZ}}\rangle$ *teleportation:* given a $|\overline{\text{CCZ}}\rangle$ resource state on logical qubits a, b, and c, a $\overline{\text{CCZ}}$ gate can be implemented on logical qubits a', b', and c' by performing three PPMs $\{\bar{Z}_a\bar{Z}_{a'}, \bar{Z}_b\bar{Z}_{b'}, \bar{Z}_c\bar{Z}_{c'}\}$, followed by three PPMs $\{\bar{X}_a, \bar{X}_b, \bar{X}_c\}$ and the corresponding $\overline{\text{CZ}}$ corrections (see Fig. 15 of Ref. [136]).

When incorporated into the above compilation scheme, Clifford circuits generally transform the processor-code logical operators appearing in these PPMs into higher-weight logical operators, eventually yielding the circuit form illustrated in Extended Data Fig. 3.
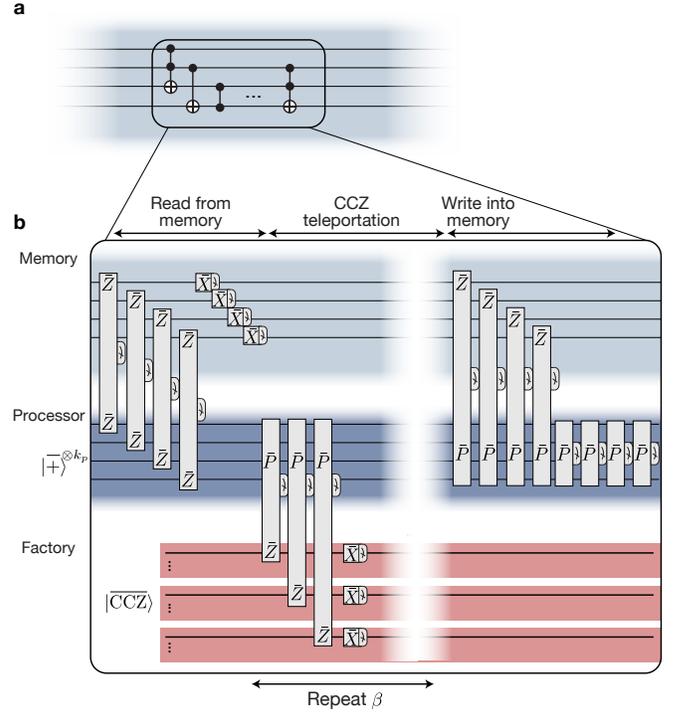
We refer to the above execution of each C_i as a *computation unit*. Each computation unit involving a subcircuit C_i thus takes time (in units of code cycles)

$$\tau(C_i) = (4m_i + 4\beta_i + \gamma_i)\tau_s, \qquad (\text{E1})$$

where τ_s again denotes a surgery cycle consisting of $\tau_s \approx 2d_p/3$ code cycles (see Sec. B).

In the above, we assumed that for each C_i the involved m_i qubits are read from the memory at the beginning and written back to the memory at the end. In other words, the processor code is initialized and measured transversally in the X basis for each computation unit. In practice, for instance, when two consecutive sub-circuits have overlapping qubit support, one could instead read only $m_i^{(\text{in})} < m_i$ qubits from the memory and write back only $m_i^{(\text{out})} < m_i$ qubits, leaving some logical qubits initialized or stored in the processor code across computation units. In this case, a similar compilation strategy applies, except that the Clifford frames must be tracked more carefully across different computation units. Importantly, since the I/O entangling operations between the memory and the processor are executed explicitly, the Clifford frame on the processor code remains unentangled from that of the memory, analogous to the situation in Ref. [37]. Consequently, the PPMs involve the large memory code (with single-qubit logical $\bar{Z}$ operators) only during the I/O operations. The time cost for such a computation unit with "partial" I/O operations is given by $\tau(C_i) = (2m_i^{(\text{in})} + 2m_i^{(\text{out})} + 4\beta_i + \gamma_i)\tau_s$. Equivalently, one may still use Eq. E1 by replacing m_i with the amortized input–output qubit count $(m_i^{(\text{in})} + m_i^{(\text{out})})/2$.

In the following section, we compute the space cost for this architecture using the concrete code instances discussed in Appendix A. Then, based on Eqs. (E1), we provide a concrete estimate for the runtime of ripple-carry adders and unary lookup tables, parameterized by

Extended Data Fig. 3. Compilation strategy. a, A general Clifford+Toffoli circuit is divided into m_i-qubit subcircuits C_i containing β_i Toffoli gates, γ_i mid-circuit Pauli measurements, and arbitrary Clifford gates. **b,** Each sub-circuit is implemented by first teleporting the m_i qubits from the memory code to the processor code. Each Toffoli gate via $|\overline{\text{CCZ}}\rangle$ teleportation and each mid-circuit Pauli measurement is implemented through sequential Clifford-transformed PPMs. Finally, the m qubits are teleported back to the memory code, again via Clifford-transformed PPMs. The PPMs can involve high-weight logical Pauli operators on the processor code, which we denote as $\overline{P}$ for simplicity.

their key parameters such as bit size. These subroutines form the core of Shor's algorithm both for ECC and RSA [18, 19, 31–37, 54, 55, 116–120].

1. Space cost

The space cost can be computed by summing the number of physical qubits in each of the four zones. We define the qubit footprint for a $[[n, k, d]]$ code to include data qubits as well as one basis (X or Z) of ancilla qubits for measuring stabilizers, i.e. $N \simeq n + \lfloor (n-k)/2 \rfloor$ (Table I). Note that in reconfigurable systems, code stabilizers can be measured in smaller batches, reducing the qubit overhead but increasing idling times. Recall that the resource zone space cost includes five bb_{18} factory blocks with parameters $[[n_f, k_f, d_f]] = [[248, 10, \leq 18]]$, and $k_f = 10$ $[[n_s, k_s, d_s]] = [[49, 1, 7]]$ cultivated surface codes. The operation zone costs are derived from Extended Data Table III, using only one basis of ancilla qubits. The resulting qubit counts are listed in Extended

Extended Data Table IV. **Space costs.** Breakdown of the physical qubit counts in the space-efficient and balanced architectures in different functional zones.

Zone	Space-efficient, $\mathsf{lp}^{3,7}_{20}$ memory	Space-efficient, $\mathsf{lp}^{3,7}_{24}$ memory	Balanced $\mathsf{lp}^{3,7}_{20}$ memory	Balanced $\mathsf{lp}^{3,7}_{24}$ memory
Memory	5,913	7,177	5,913	7,177
Processor	367	367	1,609	1,609
Resource	2,565	2,565	2,565	2,565
Operation	894	924	1,874	1,904
Total	9,739	11,033	11,961	13,255

Data Table IV.

2. Time cost for adders

Here we estimate the time costs of performing addition in the space-efficient and balanced architectures. An adder has action

$$|a\rangle |b\rangle \rightarrow |a\rangle |a + b\rangle \quad \text{(E2)}$$

on two n-bit numbers $|a\rangle = |a_1 \ldots a_n\rangle$ and $|b\rangle = |b_1 \ldots b_n\rangle$, where $|a + b\rangle$ denotes integer addition modulo 2^n [53, 107].

Because the space-efficient and balanced architectures use a small processor code block, they are best-suited for an adder circuit which operates only on a small number of logical qubits at a time. As a result, we analyze the Gidney variant [107] of the ripple-carry adder [53, 114]. Shown in Extended Data Fig. 4a, the circuit acts on three qubit registers a, b, c, where a are the offset qubits, b are the target qubits, and c are ancilla carry qubits. The circuit consists of two stages, which we refer to as the *downwards pass* and the *upwards pass* respectively. In the downwards pass, the circuit propagates the carries down from the least significant bit to the most significant bit, using Toffoli gates. In the upwards pass, the circuit uncomputes the carries using mid-circuit measurements, propagating from the most significant bit to the least significant bit. To perform q_A-bit addition, the circuit uses $3q_A$ qubits, q_A Toffoli gates, and q_A mid-circuit measurements.

Since in general the addition circuit will have too many logical qubits to fit inside the processor block ($k_p < 3q_A$), we divide the adder into smaller sub-circuits as shown in Extended Data Fig. 4. We use q_A/k_{add} computation units for the downward and upward passes respectively, where we set $k_{\text{add}} = \lfloor (k_p - 1)/3 \rfloor$. During each unit, we hold k_{add} input qubits, k_{add} output qubits, and $k_{\text{add}} + 1$ carry qubits in the processor. For each computation unit of the downward pass, we only need to read $2k_{\text{add}}$ qubits from the memory, while writing $3k_{\text{add}}$ qubits back into memory, since the carry register is initialized to zero which can be done in the processor block. Similarly, for each computation unit of the upward pass, we read $3k_{\text{add}}$

qubits but only need to write $2k_{\text{add}}$ qubits back into memory, since the carry register is reset to zero. This gives an amortized input-output qubit count of $m_i = 2.5k_{\text{add}}$. For the downward pass, each unit involves $\beta_i = k_{\text{add}}$ Toffolis and $\gamma_i = 0$ mid-circuit measurements, whilst for the upward pass $\beta_i = 0$ and $\gamma_i = k_{\text{add}}$. This leads to an overall cost of

$$\begin{aligned} \tau_{\text{adder}} &= \frac{q_A}{k_{\text{add}}} \Big((10k_{\text{add}} + 4k_{\text{add}}) + (10k_{\text{add}} + k_{\text{add}}) \Big) \tau_s \\ &= 25q_A \tau_s. \quad \text{(E3)} \end{aligned}$$

Controlled adders require an additional q_A Toffolis and q_A mid-circuit measurements on the upwards pass [107], so an analogous calculation yields an overall cost of

$$\begin{aligned} \tau_{\text{ctrl−adder}} &= \Big(\frac{q_A}{k_{\text{add}}} (10k_{\text{add}} + 4k_{\text{add}}) \\ &\quad + \frac{q_A}{k_{\text{add}}} (10k_{\text{add}} + 4k_{\text{add}} + 2k_{\text{add}}) \Big) \tau_s \\ &= 30q_A \tau_s. \quad \text{(E4)} \end{aligned}$$
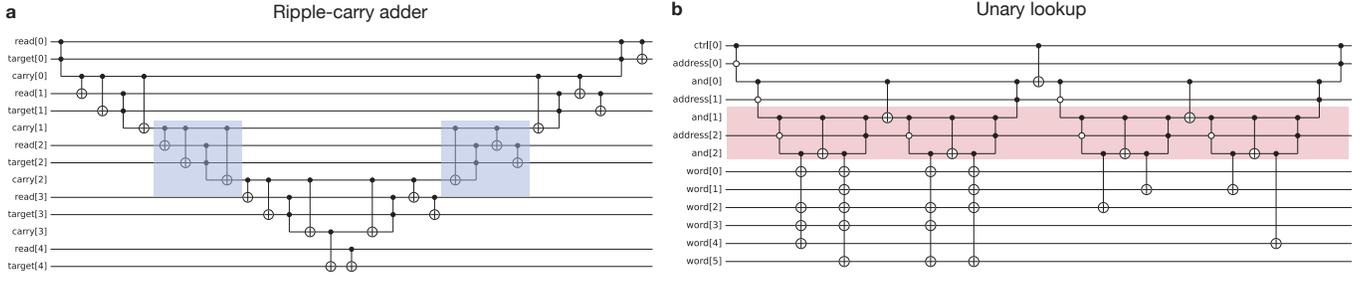
If $3q_A \le k_p$ then the entire adder fits inside the processor block. This is the case for the RSA circuit from [18] in the balanced architecture, since the maximum size of adder used is 33 bits whilst the processor has 148 qubits. In this case, we can simply teleport the entire a and b registers into the processor once at the beginning, and teleport them back once at the end. The total cost of the adder in this case is reduced to:

$$\tau_{\text{adder}} = (8q_A + 4q_A + q_A)\tau_s = 13q_A\tau_s. \quad \text{(E5)}$$

3. Time cost for lookup tables

Next we estimate the resource costs of performing lookups in the space-efficient and balanced architectures. A lookup has action

$$|\ell\rangle |\psi\rangle \rightarrow |\ell\rangle \left(\bigotimes_{i=1}^{q_w} X_i^{T_i^{(\ell)}} \right) |\psi\rangle \quad \text{(E6)}$$

Extended Data Fig. 4. **Circuits for the ripple-carry adder and unary lookup. a,** We implement the ripple-carry adder using repeated small computation units (for example the blue boxes) so that each computation unit fits inside the processor codeblock. **b,** The diagram shows an example lookup circuit on 3 address qubits and 6 word qubits. The words of the lookup table were generated randomly. Highlighted in red are three qubits: the least significant address qubit, and the two least significant ancilla AND qubits. At least half of the Toffolis and CNOTs acting on the address and ancilla AND qubits occur on these three registers, which means that at most half of the address and ancilla AND operations require I/O in our space efficient architecture.

where $\{\vec{T}^{(\ell)}\}_{\ell=0}^{2^{q_a}-1}$ is a classically-known lookup table with word length q_w [115].

We use the unary lookup circuits introduced in [115]. Shown in Extended Data Fig. 4, the circuit iterates through the address bitstrings $0, \ldots, 2^{q_a} - 1$ one-by-one. For each address bitstring, the circuit uses Toffolis to detect the identity of ℓ, before applying the $X_i^{T_i^{(\ell)}}$ operators using CNOTs. The complete circuit uses $2q_a + q_w$ qubits, 2^{q_a} Toffolis, and 2^{q_a} mid-circuit measurements [115].

If $2q_a + q_w \leq k_p$, then the entire lookup fits inside the processor, and the cost is simply

$$\begin{aligned}
\tau_{\text{lookup}} &= (4(2q_a + q_w) + 4 \cdot 2^{q_a} + 2^{q_a})\tau_s \\
&\approx (4q_w/2^{q_a} + 5)2^{q_a}\tau_s \qquad (q_w \gg q_a). \quad \text{(E7)}
\end{aligned}$$

For the balanced architecture, we assume that $2q_a < k_p$ so that the processor can store the $2q_a$ qubits needed for the unary iteration with some space to spare to apply the word operators. In this case, we will split the lookup into $\lceil q_w/(k_p - 2q_a) \rceil$ smaller lookups each involving all the address bits but a subset of at most $k_p - 2q_a$ word bits. We then implement each of the smaller lookups using an I/O step, 2^{q_a} Toffolis, and 2^{q_a} mid-circuit measurements. We only need I/O between the memory and the processor after each small lookup to teleport the completed word qubits back into the memory, since the word qubits are always initialized to zero, which can be done in the processor without I/O. As such, the whole lookup takes time

$$\begin{aligned}
\tau_{\text{lookup}} &= \left(\left\lceil \frac{q_w}{k_p - 2q_a} \right\rceil \left(2(k_p - 2q_a) + 4 \cdot 2^{q_a} + 2^{q_a}\right)\right)\tau_s \\
&\approx \frac{5q_w}{k_p - 2q_a} 2^{q_a}\tau_s \qquad (2^{q_a} \gg k_p). \quad \text{(E8)}
\end{aligned}$$

It should be noted that this compilation increases the total Toffoli count of the lookup by a factor of $\lceil q_w/(k_p - 2q_a) \rceil$, since the Toffolis on the address qubits must be repeated for each of the $\lceil q_w/(k_p - 2q_a) \rceil$ small lookups.

For the space-efficient architecture, the processor does not even have enough space to hold all of the address qubits, since $k_p < 2q_a$. In this case, we designate 3 processor qubits to hold three address qubits that control the CNOTs acting on the word qubits, as shown in Extended Data Fig. 4b. The remaining $k_p - 3$ processor qubits are used to store word qubits on which the CNOTs act. In this way, the lookup is split into $\lceil q_w/(k_p - 3) \rceil$ smaller lookups, each requiring 2^{q_a} Toffolis and 2^{q_a} mid-circuit measurements.

In this space-efficient compilation, many gates on the address qubits require I/O between processor and memory. In the complete unary iteration circuit, there are 2^{q_a} Toffolis, 2^{q_a} mid-circuit measurements, and 2^{q_a} CNOTs acting on the address qubits [115]. However, if we can hold 3 address qubits at any one time, then the structure of the unary iteration circuit implies that only half of the Toffolis, mid-circuit measurements, and CNOTs actually incur I/O cost—see Fig. 4. Moreover, each subsequent Toffoli or CNOT in the unary iteration circuit always overlaps with the previous one by at least one qubit, which means that the I/O cost is at most 2 qubits per Toffoli or mid-circuit measurement and at most 1 qubit per CNOT.

Overall, the cost of a lookup in the space-efficient architecture is:

$$\begin{aligned}
\tau_{\text{lookup}} &= \left(\left\lceil \frac{q_w}{k_p - 1} \right\rceil \left(2(k_p - 1) + 4 \cdot (2 + 2 + 1) \cdot \frac{1}{2} \cdot 2^{q_a} \right.\right. \\
&\qquad \left.\left. + 4 \cdot 2^{q_a} + 2^{q_a}\right)\right)\tau_s \\
&\approx \frac{15q_w}{k_p - 3} 2^{q_a}\tau_s \qquad (2^{q_a} \gg k_p). \quad \text{(E9)}
\end{aligned}$$

4. Time cost for RSA–2048 and ECC–256

Shor's algorithm for RSA–2048 and ECC–256 primarily uses adders and lookups as the dominant algorithmic subroutines [18, 19, 31–37, 54, 55, 116–120]. Based on the adder and lookup compilations developed in Sections 2 and 3, we now summarize and estimate the runtime of Shor's algorithm for RSA–2048 and ECC–256. We report the amortized time per Toffoli gate $\tau_{\text{Toff.}}$ for each algorithm and each architecture in terms of the surgery time τ_s; simply multiplying by the total Toffoli count will then yield the total runtime in terms of τ_s.

	RSA–2048	ECC–256
Space-efficient	$\tau_{\text{Toff.}} \approx 43\tau_s$	$\tau_{\text{Toff.}} \approx 72\tau_s$
Balanced	$\tau_{\text{Toff.}} \approx 10\tau_s$	$\tau_{\text{Toff.}} \approx 19\tau_s$

Extended Data Table V. Amortized time-per-Toffoli for the RSA–2048 and ECC–256 algorithms in the space-efficient and balanced architectures, all in terms of the time τ_s of a single surgery operation.

For the RSA–2048 circuit in Ref. [18], approximately 50% of the Toffolis arise from lookups on roughly 6 address bits with word-size at most 33, and the remaining 50% of Toffolis arise from adders on at most 33 bits [18]. In the space-efficient architecture with $k_p = 10$, the time-per-Toffoli for adders and lookups is approximately $25\tau_s$ and $(15q_w/(k_p - 3))\tau_s \approx 71\tau_s$ respectively. Overall this yields:

$$\tau_{\text{Toff.}} \approx 0.5 \cdot 25\tau_s + 0.5 \cdot 71\tau_s \approx 43\tau_s \qquad (\text{E10})$$

for RSA–2048 in the space-efficient architecture.

In the balanced architecture with $k_p = 148$, the small adders and lookups used in RSA–2048 fit inside the processor, so the time-per-Toffoli for adders and lookups is only $13\tau_s$ and $(4q_w/2^{q_a} + 5)\tau_s \approx 7\tau_s$ respectively. This yields:

$$\tau_{\text{Toff.}} \approx 0.5 \cdot 13\tau_s + 0.5 \cdot 7\tau_s \approx 10\tau_s \qquad (\text{E11})$$

for RSA–2048 in the balanced architecture.

For the ECC–256 algorithm, based on compilations in [116–120] we assume the following split in the Toffoli count between controlled-adders, adders and lookups:

- 40% 256-bit adders,
- 50% 256-bit controlled-adders,
- 10% lookups with 16 address bits and word size 256.

In the space-efficient architecture with $k_p = 10$, the time-per-Toffoli for adders and controlled-adders is $25\tau_s$ and $15\tau_s$ respectively, and the time-per-Toffoli for lookups becomes $(15q_w/(k_p - 3))\tau_s \approx 550\tau_s$. Overall, for ECC–256 in the space-efficient architecture we get:

$$\tau_{\text{Toff.}} \approx 0.4 \cdot 25\tau_s + 0.5 \cdot 15\tau_s + 0.1 \cdot 550\tau_s \approx 72\tau_s \quad (\text{E12})$$

In the balanced architecture with $k_p = 148$, the adders and controlled adders again have time-per-Toffoli $25\tau_s$ and $15\tau_s$ respectively, but now the time-per-Toffoli for lookups is $(5q_w/(k_p - 2q_a))\tau_s \approx 11\tau_s$. Overall, for ECC–256 in the balanced architecture we get:

$$\tau_{\text{Tof}} \approx 0.4 \cdot 25\tau_s + 0.5 \cdot 15\tau_s + 0.1 \cdot 11\tau_s \approx 19\tau_s \quad (\text{E13})$$

Appendix F: Time-efficient architecture

Here we describe the time-efficient architecture and its associated resource costs for RSA–2048 and ECC–256. This architecture parallelizes Toffoli gates in core algorithmic subroutines, which can reduce time costs compared to the serial space-efficient and balanced architectures (Appendix E). As a proxy for the time reduction relative to these serial architectures, we consider replacing linear-depth ripple-carry addition [53] used in prior resource estimate circuits [18, 34, 54, 116–120] with logarithmic-depth quantum carry-lookahead addition [123]. Similar linear-to-logarithmic-depth reductions are also possible in lookup tables [159], which together with adders comprise the dominant Toffoli counts in prior circuits for Shor's algorithm [18, 19, 31–37, 54, 116–120] (see Appendix E). As a result, we expect that the speedup between the ripple-carry and carry-lookahead adder is a reasonable proxy for the speedup of the full circuit.

To leverage these lower-depth primitives, we generate and consume magic $|\overline{\text{CCZ}}\rangle$ states in parallel using the high-rate $8T$-to-CCZ distillation protocol (Appendix C), extended to larger factory codes encoding more logical qubits. In addition, we assume logical gadgets capable of measuring many logically disjoint PPMs on high-rate codes in parallel. Such capabilities have been demonstrated using, e.g. high-rate surgery techniques [52, 121, 122] (see also Appendix B), achieving parallel measurements of up to hundreds of logical operators on distance-~ 10 codes with ancilla overhead on the order of 1–$2\times$ the code size [121]. Ongoing work further develops these constructions for LP codes [113]. We characterize variants of the time-efficient architecture by the *parallelism* P, defined as the number of $|\overline{\text{CCZ}}\rangle$ states distilled and consumed simultaneously.

Our goal is not to specify concrete circuits or instruction sets, but to provide an approximate resource estimate for architectures supporting highly parallel Toffoli gates. The schemes outlined below rely on continued improvements in such parallel logical gadgets, which we expect to arise from further optimized high-rate surgery techniques [52, 79, 113, 127] or alternative approaches such as homomorphic measurements [51, 146]. With this goal in mind, we first outline the architecture at a high level and then estimate its resource costs.

1. Description

In our time-efficient architecture, we perform logical operations directly on code blocks used for processing, thereby avoiding costly communication overhead with the memory blocks. Upcoming work finds codes at sufficiently high distance hosting $\gtrsim 100$ logical qubits with 20% encoding rates, and $\gtrsim 600$ logical qubits with 30% encoding rates [125], which we leverage for processing and as factory code blocks. We generate P magic $|\overline{\mathrm{CCZ}}\rangle$ states at a time using high-rate distillation with five factory code blocks and cultivated surface codes. As described later, we use different codes for processing and magic depending on the chosen value of P.

Using the carry-lookahead adder described in Ref. [123], we consider both controlled and uncontrolled adders, which can each appear in cryptographic algorithms, depending on the particular circuit [18, 34, 54, 116–120]. In both cases, the circuits are comprised of $\approx 4 \log(n)$ parallel layers of Toffoli gates, each acting on a potentially different subset of logical qubits. This can be contrasted with ripple-carry addition, which requires $\approx 1n$ and $\approx 2n$ Toffoli layers for uncontrolled and controlled addition respectively. The Clifford cost for the carry-lookahead adder is minimal—at most six layers of CNOTs in the entire adder—and therefore we neglect it in our estimates.

The carry-lookahead circuit [123] proceeds by first generating a group of P magic states using the high-rate distillation described in Appendix C. At large P, the space cost from surface codes can become large, so we allow the surface codes to be injected into the $\overline{T}$ block in a variable number of batches n_{batch}, resulting in a time cost per distillation of $8 \times 15 n_{\mathrm{batch}}$ (see Appendix C). The resulting magic states are teleported into the computation using a single layer of parallel $\overline{ZZ}$ PPMs between the factory codes and the processor codes. Subsequent $\overline{CZ}$ fix-ups on the processor codes are performed using two layers of parallel $\overline{ZZ}$ or $\overline{ZX}$ measurements with the assistance of up to $3P$ ancilla logical qubits [124], which could come from e.g. the factory codes that are freed after the $|\overline{\mathrm{CCZ}}\rangle$ teleportation. Assuming each layer of parallel two-body PPMs can be implemented in parallel using, e.g. high-rate surgery, the total time cost of gate teleportation and fixups is therefore $3\tau_s = 2d$, where d is taken to be 20 and τ_s denotes the surgery cycle time. To reduce time costs, for layers involving fewer than five Toffoli gates, we delay the fixups until the next layer involving more than five Toffoli gates, at which point they are performed at negligible increase to the surgery system size. At each step, the maximum possible number of Toffolis are implemented, corresponding to either P (consuming a full factory) or the number of remaining Toffoli gates in the current Toffoli layer, whichever quantity is smaller. As soon as the produced $|\overline{\mathrm{CCZ}}\rangle$ states are fully consumed, another batch is produced and the computation proceeds.

2. Resource costs

We now estimate the space and time costs of this procedure for ECC–256 and RSA–2048 at different levels of parallelism. The space cost is estimated from the sum of three quantities: the processor size, the resource zone size, and the operation zone size. As with the other architectures, our total qubit count includes data qubits and stabilizers in one basis (X or Z); see Extended Data Table I. To estimate the processor size, we first compute the total number of logical qubits which need to be stored, given by the sum of the number of logical qubits from the original compilation [34, 55] and the number of ancilla logical qubits required for carry-lookahead addition, $n - 2\log(n)$, where n is the number of bits in the adder [123]. We then divide by the encoding rate r for the chosen P to estimate the corresponding number of processor data qubits, from which we compute the total qubit count. For $P < 600$ ($P \geq 600$), we assume processor encoding rates of $r = 20\%$ ($r = 30\%$) based on upcoming work [125]. Note that because we do not specify the concrete block size, these numbers are correct up to rounding errors of one block, which are small compared to the total qubit count.

For the resource zone, we require five factory blocks each encoding P logical qubits which can be transversally coupled (note that each factory block can be assembled from smaller, independent code blocks). We assume these codes have the same rate as the processor, with the exception of $P < 100$, where we assume a lower encoding rate of 4% to allow for transversal coupling of smaller block sizes (e.g., the bb_{18} code). We also require P/n_{batch} surface codes for $|\overline{T}\rangle$ cultivation. Finally, we estimate the operation zone size by noting that at most $6P$ logical qubits are operated on in parallel at once (corresponding to three blocks for the $|\overline{\mathrm{CCZ}}\rangle$ states, and $3P$ processor logical qubits).

We estimate the required ancilla size as $\gamma(6P/r)$, where γ is the ratio between the ancilla size for measuring a layer of parallel PPMs and the size of the corresponding codes. For reference, $\gamma \approx 1$–2 for $d \sim 10$ codes with X- or Z-type measurements using high-rate surgery [52]. To account for larger codes and more complex PPMs, we consider $\gamma = 1$–3 and plot results with $\gamma = 2$ in Fig. 3. We further assume that the ancilla cost for injecting $|\overline{T}\rangle$ states during distillation does not exceed this bound. Because some physical qubits are freed up after the distillation factory terminates (including all of the surface code qubits and two factory blocks), these extra qubits are repurposed for the operation zone for teleportations and fixups, further reducing its total size. We re-emphasize that the operation zone size is an estimate, and future work can benchmark and optimize logical gadgets for measuring parallel PPMs.

Now we estimate the space and time costs for a parallelism level P. For RSA–2048 we consider $P = 100$ and $P = 1,160$ for which we choose $n_{\mathrm{batch}} = 2$. For ECC–256 we consider $P = 20$ and $P = 130$, for which

we choose $n_{\text{batch}} = 1$. In the main text, we assume $\gamma = 2$. We estimate the range in space costs due to surgery system size fluctuations by considering $\gamma = 1$ and $\gamma = 3$. This corresponds to approximate uncertainties of $67{,}000^{+4{,}000}_{-3{,}000}$ qubits ($P = 100$) and $102{,}000^{+28{,}000}_{-4{,}000}$ qubits ($P = 1{,}160$) for RSA–2048, and $19{,}000^{+3{,}000}_{-2{,}000}$ ($P = 20$) and $26{,}000^{+5{,}000}_{0}$ qubits ($P = 130$) for ECC–256. (The lower bound for the final estimate is zero because the space is dominated by the factory rather than code surgery.) We compute the factor speedup compared to the balanced architecture by comparing to the time cost of the ripple-carry adder, given by $\frac{2d}{3} \cdot 25n$ and $\frac{2d}{3} \cdot 2 \cdot 15n$ for non-controlled and controlled addition, respectively. We take the ratio of these time estimates with the depth of the carry-lookahead adder, and take the average of the speedups for both controlled and non-controlled adders. The associated space and time costs are plotted in Fig. 3b-c in the main text.

Competing interests The authors are shareholders of Oratomic, Inc., which is developing fault-tolerant quantum computers. M.C., Q.X, R.K., L.R.B.P, H-Y.H, and D.B. are full-time employees, and H.L., M.E., and J.P. are part-time employees, of Oratomic, Inc.

Correspondence and requests for materials should be addressed to M.C., Q.X, and D.B.