Superconducting qubits in the millions: The potential and limitations of modularity

S.N. Saadatmand,[1,*] Tyler L. Wilson,[1] Mark J. Hodson,[1] Mark Field,[1] Simon J. Devitt,[2,3] Madhav Krishnan Vijayan,[3] Alan Robertson,[3] Thinh P. Le,[3] Jannis Ruh,[3] Alexandru Paler,[4,2] Arshpreet Singh Maan,[4,2] Ioana Moflic,[4,2] Athena Caesura,[5] and Josh Y. Mutus[1]

[1] *Rigetti Computing, 775 Heinz Avenue, Berkeley, California 94710, USA*

[2] *InstituteQ, 02150 Espoo, Finland*

[3] *Centre for Quantum Software and Information, University of Technology Sydney, Sydney, New South Wales 2007, Australia*

[4] *Aalto University, 02150 Espoo, Finland*

[5] *Zapata AI Inc., Boston, Massachusetts 02110, USA*

(Received 11 August 2025; revised 9 March 2026; accepted 22 April 2026; published 9 June 2026)

The development of fault-tolerant quantum computers (FTQCs) is receiving increasing attention within the quantum computing community. Like conventional digital computers, FTQCs, which utilize error correction and millions of physical qubits, have the potential to address some of humanity's grand challenges. However, accurate estimates of the tangible scale of future FTQCs, based on transparent assumptions, are uncommon. How many physical qubits are necessary to solve a practical problem intractable for classical hardware? What costs arise from distributing quantum computation across multiple machines? This paper presents an architectural model of a potential FTQC based on superconducting qubits, divided into discrete modules and interconnected via coherent links. We employ a resource-estimation framework and software tool to assess the physical resources required to execute specific quantum algorithms compiled into their graph-state form and arranged onto a modular superconducting hardware architecture. Our tool can predict the size, power consumption, and execution time of these algorithms based on explicit assumptions about the physical layout, thermal load, and modular connectivity of the system. We assess the resources needed for quantum computation examples that serve as building blocks of proposed applications, quantifying the architectural bottlenecks and trade-offs that remain to be addressed to deliver utility.

DOI: 10.1103/k3d5-v43c

I. INTRODUCTION

Recent studies suggest that future idealized FTQCs with $(1–10) \times 10^6$ physical qubits can address classically intractable problems in integer factoring [1] and molecular chemistry [2,3], completing their assigned tasks in a matter of days. At the same time, the size, connectivity, and fidelity of today's noisy physical-qubit devices are continuously improving, and now support some of the fundamental building blocks of fault tolerance [4–6]. These emerging platforms employ quantum error-correction (QEC) codes [6–13] to reduce logical errors at the cost of additional resources: more space, meaning physical qubits, and more time needed to operate the code.

The complexity of fault-tolerant (FT) algorithms, including Shor's factoring [14], quantum signal processing [15] (QSP), and ground-state energy estimation [16], is well understood. However, research on the magnitude and scaling of the physical resources required to execute such computations is limited. Furthermore, the overhead and space-time resource requirements for physical hardware are even less well understood and depend on platform and microarchitecture choices. Scaling tangible resources such as energy, space, and time, based on specific assumptions about a potential machine, is crucial for evaluating the feasibility of methods for developing FTQC.

Fault-tolerant resource-estimation tools currently exist, but are restricted to using the "T-counting" method [10,17–20] only. These footprint-style estimation tools synthesize the logical circuit into Clifford+T gates (or other universal gate sets), rescale the space-time volume according to the desired level of parallelization, and derive physical

*Contact author: nsaadatmand@rigetti.com

space-time quantities based on the T count and circuit depth, along with assumptions regarding T distillation and hardware timescales. This methodology lacks targeted estimates specific to quantum processing unit (QPU) layouts, provides insufficient estimates for surface-code distance, and may overestimate the upper limits of space-time resources.

Here, we present a transparent FT architectural model for square-lattice superconducting-qubit systems that uses the surface code [7,10,21–23]. This model supports the essential elements of FTQC: logical Clifford gates via lattice-surgery-based approaches [10,24–27], magic or T-state distillation (MSD) [9,10,28] for non-Clifford gates, and queuing and routing of specific resources. We also establish explicit assumptions about the size of a single module and the penalties for executing an FT algorithm that exceeds its capacity, allowing us to quantify the impact of coherently distributing a quantum computation.

We also present application-specific resource estimates for the proposed FT superconducting-qubit architecture. These estimates are derived from a graph-state method detailed in Sec. II A. This framework aims to assist the design and feasibility assessment of next-generation FTQCs while quantifying the trade-offs associated with hardware implementations. We also discuss an accompanying open-source software called Rigetti Resource Estimations [29] (RRE), which we used to generate all the results presented in Sec. IV for a selection of test algorithms outlined in Sec. III. We will explore potential future work in Sec. V and provide a conclusion in Sec. VI.

II. A PRIMER FOR ALGORITHM EXECUTION, ARCHITECTURE DESIGNS, AND RESOURCE ESTIMATION ON FTQCs

The execution of algorithms on an FTQC fundamentally differs from merely executing the gates of a quantum circuit in the order in which they appear. By definition, FTQCs depend on the underlying QEC scheme they use (see, e.g., Refs. [6,7,9,10,12,13]). FTQCs employ noisy physical qubits, encoded within an error-correction scheme, to create logical qubits with exponentially suppressed error rates.

The error-correcting scheme we consider here is the surface code [7,10,21,22], implemented on square arrays of physical qubits. In an array, physical qubits are categorized as *data* or *ancilla* qubits, with each ancilla qubit coupled to four data qubits and vice versa. Physical ancilla qubits enable repeated composite parity measurements on their four neighboring data qubits in the form $X_1X_2X_3X_4$ or $Z_1Z_2Z_3Z_4$. These composite parity measurements are performed by entangling the physical data qubits with the ancilla via controlled-NOT (CNOT) gates and measuring the ancilla in either the Pauli X or Z basis. Physical errors on the physical qubits in these arrays result in patterns of bit flips among the parity measurements, known as *syndromes*. Tracking the outcomes of the parity measurements and the resulting error syndromes, and mapping them to the most probable error event, is known as *decoding*. In practice, corrections can be applied after decoding through judicious syndrome tracking and software corrections after running the algorithm (potentially incurring time delays due to decoder latency).

One price we pay for this encoding is that the resulting logical qubits can perform only a limited set of FT operations; specifically, the Clifford set. At least one non-Clifford operation must be engineered into the FTQC to achieve universality and execute arbitrary algorithms fault tolerantly. We consider the T gates (a $\pi/4R_z$ rotation) as the non-Clifford operations choice in this paper. However, forcing an error-correcting code to produce an operation outside its native subspace (such as a T gate) involves significant overhead, necessitating the dedication of substantial portions of the FTQC to MSD or, more recently, cultivation [9,10,28] to produce T gates.

Due to the constraints imposed by the error-correcting code, we ultimately decompose the algorithm into a sequence of Clifford + T gates. This separation of functional operations leads to a specialization of the fundamental architecture: certain sections of the FTQC are dedicated solely to performing the original logical Clifford operations through multi-product (-qubit) Pauli measurement operations or MPPOs—a process based on the lattice surgery [10,24–27]. For example, MPPOs can be understood as applying n-body controlled-Z (CZ) gates and single-qubit Clifford gates to the quantum state, thereby leading to a formalism that we later explore. Additionally, there are designated areas for generating and queuing T states produced via MSD. Our proposed logical layout for this operational model is presented schematically in Fig. 1.

We adapt this logical-operation model and the fixed yet extensible microarchitecture of Fig. 1 to a physical layout of superconducting qubits with square-lattice connectivity. We estimate the execution of selected algorithms on this architecture. To this end, we generate schedules of quantum operations using a graph-state-based approach, as proposed in Refs. [30–37] and further refined below. This approach yields several abstractions that facilitate distributing the algorithm across fixed hardware. Specifically, we can manipulate graphs or subgraphs to tailor the overall execution of the algorithm within an FTQC composed of distinct yet interconnected modules.

One can implement the original quantum algorithm using the algorithm-specific graph (ASG) formalism [32,38,39] and associated schedules of operations [33,34], which apply only logical measurements and classical feedforward. The ASG formalism is a resource-efficient variant of the measurement-based quantum computing (MBQC) framework [35,40]. MBQC is particularly appealing for distributed FTQC because it requires only an

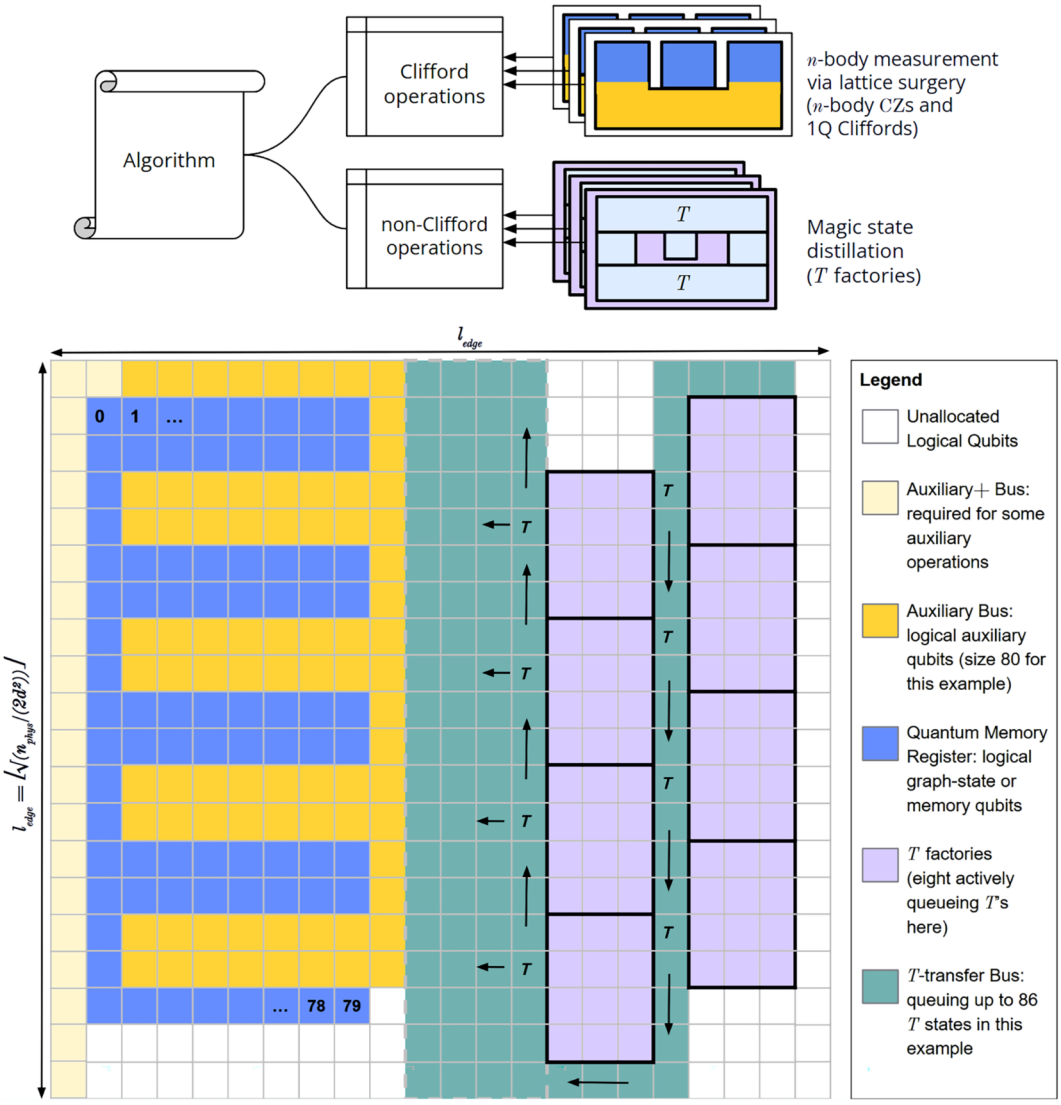

FIG. 1. (a) Fault-tolerant algorithm execution involves segmenting the application into logical Clifford and non-Clifford portions. Clifford operations are performed using a set of measurements on logical memory qubits, facilitated by auxiliary buses. Because measurement outcomes are probabilistic, classical feedback and feedforward are incorporated into operation schedules. Non-Clifford portions are implemented using T states distilled by T factories. (b) This results in our proposed logical microarchitecture, detailed in Sec. II C. The architecture includes graph preparation, consumption, local T factories, and T-state and Bell-state queuing, with some parallelization. We arrange logical qubits on a square lattice of rotated surface-code patches of distance d made of physical qubits. A left portion of the module is allocated for the logical memory (blue tiles). It also contains an "auxiliary bus" of logical auxiliary qubits needed for graph preparation, consumption, T, and Bell-state routing. Yellow tiles represent the auxiliary bus, followed by a lighter-yellow "auxiliary+ bus" required for routing. To its left, a separate bus is designated for continuous queuing of T states distilled in an array of T factories on the opposite side of the module (teal tiles)—see also parameters 6 and 8 of Appendix A. This architecture can be distributed across multiple modules, provided that they are coherently connected via the auxiliary bus and each module consumes local T states.

algorithm-specific resource, such as the graph states and measurements on logical qubits. We will explore and utilize a specific form of MBQC as outlined in Ref. [32] for compilation purposes. We prepare the graph-state resource using lattice-surgery-based processes to entangle logical qubits in a memory register, facilitated by a single bus of logical auxiliary qubits. Due to the probabilistic nature of quantum measurements, we must track the set of measurement results classically using a process called Pauli frame tracking [33,41–43]. Next, we move locally distilled T states, through patch deformations [10,24], to the register of logical memory qubits to apply the non-Clifford portion in conjunction with the Pauli frame corrections of the algorithm. Finally, we apply any corrections flagged

by syndromes tracked by the decoder, resulting from physical errors in the hardware. We present a mathematical description of the ASG formalism in Sec. II A.

We need to arrange the elements of the graph-state-based algorithm execution across physical components at scale. In large-scale classical computations, there are limits to the amount of computation that can be performed on a single processor within a given time frame. These limits stem from the processing power that can be integrated into a single chip, ultimately constrained by physical limits and manufacturing yields. Consequently, computations are often distributed across multiple processors and machines. This distribution incurs penalties, such as higher latency or reduced bandwidth, but it enables greater overall computational power. We can draw an analogy with distributed FTQC, which involves a network of interconnected compute modules. Manufacturing yields limit the quantum computing power of each module. These operational (thermal) and manufacturing constraints restrict the number of physical qubits that can be incorporated into a single QPU housed within a refrigeration module. This has prompted us to propose a logical macroarchitecture composed of several modules, as described in Sec. II B.

In contrast to the flexible connectivity and relatively slow speeds between compute modules, QPUs based on superconducting qubits offer fixed connectivity and faster gate speeds. In Sec. II C, we describe the hardware microarchitecture, explain how it can be customized to accommodate specific hardware constraints, and provide a physical rationale for fixing the number of physical qubits in a single compute module to be around one million. We propose sparsely and coherently interconnecting the modules, which may result in reduced fidelity, lower bandwidth, or slower entangling gate speeds. We will refer to the "coherent interconnect" between modules with the surface-code distance of d as a single "pipe." We will abstract the transduction efficiency as an interaction time, characterized by the "intermodule tock," and quantify how this time impacts the overall run-time of the algorithm. We assume that the pipes share identical noise characteristics and code distance with the rest of FTQC.

We present our compilation and estimation methodology for RRE, based on user inputs, in Sec. II D 7. Further details on the RRE output glossary, our circuit decomposition strategy, and the verification of RRE outputs are provided separately in Appendices A, B, and C, respectively.

Setting aside the technical details of our proposed logical layouts, it is important to note that the fixed intramodule microarchitecture in Fig. 1 and the intermodule macroarchitecture in Sec. II B were not tailored to any specific application. Instead, we adapted the overall architecture to the strengths and limitations of our superconducting hardware and the graph-state-based algorithm execution. We will then evaluate its performance on selected applications. In Fig. 1, the bilinear pattern in the memory and auxiliary buses arises from the ease of scheduling the full FT algorithm using graph states. T factories are kept local to modules and connected via another bus primarily because practical algorithms require a large number of T operations, and coherent interconnects are slow for T teleportations. However, we assume that coherent interconnects are effective for less frequent Bell-state-based teleportation of the graph state. This is why we propose the distributed macroarchitecture described in Sec. II B; the graph state would not fit into a single module for practical algorithms. Future design improvements could include alternative memory and bus structures, such as city-style patch layouts and more efficient distinct QEC codes for memories, to reduce reliance on many coherent interconnects, which may be more suitable for certain applications, as suggested by our results in Sec. IV.

A. Mathematical description of the graph-state formalism

A mathematical graph, denoted by $\mathcal{G}$, is defined by its vertices and edges. Given a graph, $\mathcal{G}$, the associated quantum (Clifford) state, called a *graph state*, is constructed by assigning each vertex in $\mathcal{G}$ to a logical qubit initialized to $|+\rangle$ and each edge in $\mathcal{G}$ to an entangling CZ operation that links the logical qubits at the corresponding vertices. We often refer to the logical qubit associated with a vertex as a "node" and use "graph" and "graph state" interchangeably for brevity. The nodes undergo non-Pauli-basis measurements corresponding to the non-Clifford gates of the original circuit and some X-basis measurements to facilitate teleporting unknown inputs into the graph state [32].

The ASG formalism provides measurement schedules for two main stages to fully execute the FT algorithm: one schedule details the preparation of the graph through lattice surgery [corresponding to the original nontrivial Cliffords, first stage, in Fig. 1(*top*)] and another outlines the necessary single-qubit measurements in non-Clifford bases [corresponding to the original non-Cliffords, second stage, in Fig. 1(*top*)]. We also need to track Pauli frame corrections in the software. We denote the nested set of stabilizer-measurement schedules that "prepare" the graph in the first stage as $S_{\mathcal{G}}^{\text{prep}}$ and refer to it as the preparation schedule for the graph state. $S_{\mathcal{G}}^{\text{prep}}$ specifies how to initialize $\mathcal{G}$, the graph state, based on the original entangling operations and other step-by-step requirements [34,44]. Conversely, we denote the nested set of non-Clifford measurement schedules that "consume" the graph state in the second stage as $S_{\mathcal{G}}^{\text{consump}}$ and refer to it as the consumption schedule. $S_{\mathcal{G}}^{\text{consump}}$ specifies how to perform logical $R_Z(\theta)$-basis measurements (for arbitrary non-Clifford angle θ) relevant to executing the original non-Cliffords [33,45,46]. Both schedules consist of subsets or subschedules that specify the order of

measurements, with each subset containing the nodes designated for simultaneous measurements [32,44,45]. Furthermore, the essential stages of MSD, routing T states, and decoding can occur at either stage or be postponed, depending on the requirements.

At a high level, the ASG formalism requires the FT algorithm to be ingested as single- and two-qubit gates. We employ a straightforward method for decomposing, compiling, and executing the algorithm, segmenting it into equal-width recurring subcircuits along the time axis. These subcircuits are called "widgets," have their own (sub)graphs, and reconstruct the original algorithm when "stitched" together (see also Sec. II D 7). We refer to this decomposition process as "widgetization," detailed in Appendix B. We will examine the graph properties of these widgets and the scheduling process for preparing and consuming them on the proposed hardware architecture.

B. High-level macroarchitecture for modular graph-state processing

We consider a macroarchitectural model that, at its core, uses superconducting qubits on square lattices and the logical layout shown in Fig. 1(bottom). This enables us to estimate the resources required to execute the FT algorithm and understand the associated trade-offs. To this end, we consider a physical layout for FTQC consisting of sets of *rotated* surface-code patches, which are tiled together to create all intramodule components. For example, a group of patches forms a graph-state memory register, while others are designated for the T-transfer bus. We assume that each patch contains $2d^2$ physical qubits. Although the rotated surface code requires only $2d^2 - 1$ physical qubits [10,21–23], we adopt a physical size of $2d^2$ for all patches to streamline our approximate scaling calculations, ignoring nondominant spatial terms and tiling details.

Executing the quantum algorithm based on the ASG formalism and time-direction widgetization inherently yields a macroarchitecture that partitions the hardware into sets of modules that execute widgets in an interleaved fashion. The question is how to lay out modules and assign them to potentially interleaving operations. In steady state, the execution of the quantum algorithm comprises *three* main steps: (1) preparation or creation of the graph, which initializes a quantum state using the original nontrivial Clifford and entangling operations; (2) consumption of the graph, which executes the original non-Cliffords; and (3) following every consumption step, the handover of the graph through Bell-pair teleportations or by stitching output nodes from the preceding widget to the input nodes of the current widget.

Step 2 for the current widget and step 1 for the widget can occur concurrently during graph-state processing. This interleaved execution structure enables us to explore a degree of parallelism in a distributed FTQC. We propose distributing the interleaving preparation and consumption steps across two physically distinct sets of modules positioned along the two legs of a ladder structure. Teleportation may be used to mediate graph-processing operations spanning multiple modules, but these should be minimized in scheduling. The graph-processing cycle is streamlined because each set of modules alternates between only preparing or consuming the graph at each step. In contrast, the other sets of modules do the opposite on the adjacent widget, as illustrated in Fig. 2. During all preparation and consumption steps, the FTQC performs MSD within the modules and fills a dedicated T-transfer bus as required. The timescales for T-state distillation and intermodule operations are typically one order of magnitude worse than those for other surface-code operations, including graph preparations, so MSD remains local to individual modules to improve computation speed. Only the consumption stage requires the FTQC to apply the T states; since graph preparations and MSD are performed independently, we need to account for possible preparation, MSD, and other delays in the consumption stage of each widget. For the consumption steps, some graph-state nodes require a single T-basis measurement, while others require multiple T-basis measurements to execute non-Clifford rotations in the original algorithm. Subsequently, the FTQC teleports the current graph's output nodes to the input modes of the newly prepared widget. This cycle continues until the FTQC processes all widgets in the complete algorithm (see also Sec. II D 7).

The architecture must be scalable to accommodate algorithms that require more space-time resources than a single module or time step. Furthermore, MSD is arguably the most resource-intensive component of a universal FTQC, and we assume that each module has an equal number of T factories supplying T states locally. To address both issues, we propose a ladder-style macroarchitecture illustrated in Fig. 3 (see Fig. 1 for the intramodule layout). The quantum memory register will be shared equally among all modules on the same ladder leg at any given time. Here, we can use the faster native connectivity of a lattice of superconducting qubits within a module for graph consumption and MSD, while reduced connectivity or slower interactions are used for some graph-preparation operations and for teleporting the output nodes of graph states on one leg of the ladder to the input nodes on the other leg.

We can estimate the time required to execute the system-wide operations schedule by accounting for all local and intermodule operations. To account for intermodule operations, we abstract the characteristics of the teleportation interaction between modules into a single time, referred to as the *intermodule handover time*, and set it to $t_{\text{inter}} = 1\ \mu\text{s}$. We incorporate this characteristic time to measure the cost of scheduled operations that cross a module boundary and to estimate the total FT handover time.

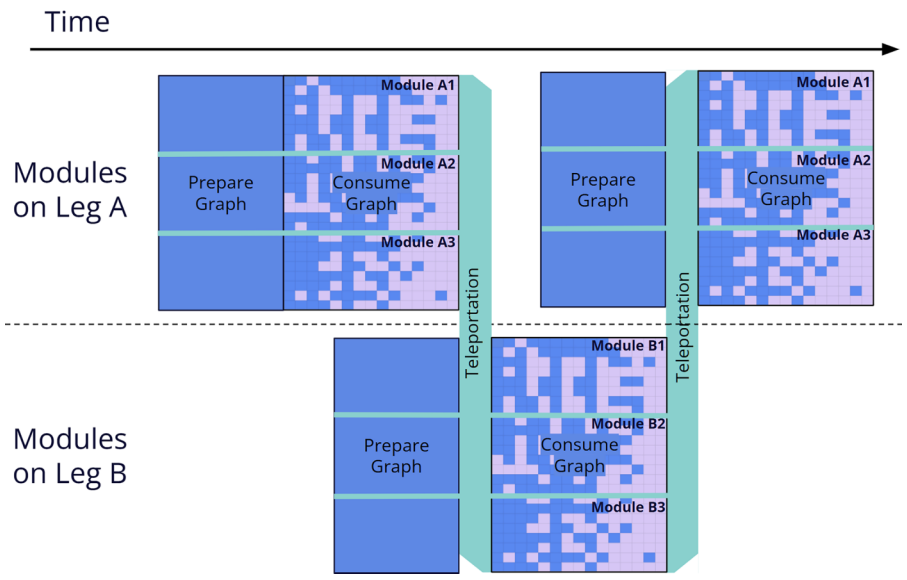

FIG. 2. The high-level graph-state processing cycle: the algorithm is divided into time-sliced widgets that are interleaved across two module sets in the time direction. Each set, consisting of three modules in this example, interleaves graph-state preparation with graph-state consumption. Graph-state preparation consists solely of Clifford operations (blue), while consumption utilizes T-state resources (purple). Teleportation moves logical state between module sets, from the output of one graph-state consumption to the input of the next. Teleportation is also used to facilitate operations that cross module boundaries within the set.

C. Superconducting hardware microarchitecture and lower-level components

We propose a logical microarchitecture concept (see Fig. 1) that informs a system-level model, which, at a minimum, must include the following components:

(1) At its core, a lattice of rotated surface-code physical qubits, used as logical patches to build the following distinct subsystems.
(2) A register of logical graph-state or memory qubits for graph preparation and consumption.
(3) Continuous buses of logical auxiliary qubits that couple with, connect to, and operate on logical memory qubits while routing T states for graph consumption.
(4) Another continuous bus of logical auxiliary qubits for queuing and moving T states that must maintain a broad enough connection to the preceding auxiliary qubits to transfer T states to them on demand.
(5) T factories.
(6) Quantum coherent interconnects between modules to enable connections along the auxiliary bus portions.

Section II highlighted the critical mechanism that executes quantum algorithms in the ASG formalism: a series of non-Clifford rotations and MPPOs on logical graph-state qubits, supported by auxiliary buses, perform logical operations. Consequently, this architecture requires cryostat modules to connect solely via the buses of logical auxiliary qubits. This significantly simplifies the hardware, as dense long-range connectivity between modules is no longer necessary. The essential requirement is that the number of coherent interconnects along the buses must equal the code distance d; adding additional connections will enhance overall computation for specific applications. We will quantify this trade-off in Sec. IV.

Here, we outline the physical specifications of the system to ensure that the necessary resources to execute the FT algorithm are available. We start with the physical-qubit layout. To provide the most relevant estimates for current hardware, we base our projections on a superconducting-qubit architecture consisting of tunable transmon qubits coupled via tunable couplers [47]. This architecture has been implemented with high fidelity by several groups [48–51]. Here, we lay out the superconducting transmon qubits in a square lattice, coupled to one another via tunable couplers. This square lattice directly corresponds to the surface-code patches, allowing us to relate physical-qubit numbers to the code distance easily.

To determine the number of physical qubits per module and the overall size of the QPU, we must assess the qubit pitch. In this tunable coupled-transmon architecture, the unit cell consists of a transmon and two tunable couplers. Accounting for this, each physical qubit requires 3.1 control signals at the chip, allocated to:

(a) $1\times$ qubit control (flux bias for frequency tuning and rf drive for gates, combined externally to the chip via a diplexer [52]);
(b) $2\times$ coupler bias lines; and

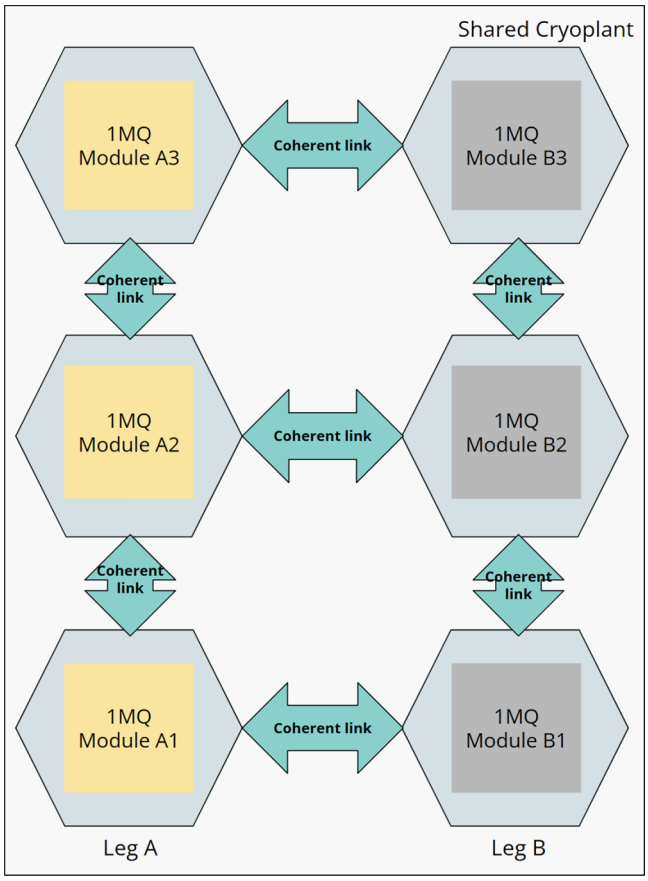

FIG. 3. The high-level macroarchitecture consists of two sets of 10^6-physical-qubit (1MQ) modules arranged like the legs of a ladder. Following the example in Fig. 2, each leg contains three modules, each following the layout of Fig. 1(bottom). Portions of the algorithm are divided and executed solely within modules on the same leg, where intramodule interactions with high connectivity and short gate times dominate. All modules communicate via width-d "coherent interconnects" (or links) at a lower rate and with reduced connectivity, used only for graph preparation (vertical interconnects) and state handover (horizontal interconnects).

(c) $0.1\times$ rf line for readout resonators shared among ten qubits for multiplexed dispersive readout [53].

While most QPUs employ perimeter wire bonding for individual control lines, several proposals and demonstrations focus on addressing physical qubits individually using through-substrate vias (TSVs) [54,55].

By adopting the TSVs method, QPUs require one signal pin per physical qubit and one per tunable coupler. Each signal line needs at least one ground line for return currents. As schematically illustrated in Fig. 4, achieving a sufficient number of signal and ground connections is possible if the physical-qubit pitch P_q is three times the via pitch P_v. Ultimately, the transmon, coupler, and TSVs can all be fabricated with a pitch of less than 100 μm. However, the bottleneck occurs at the input and output:

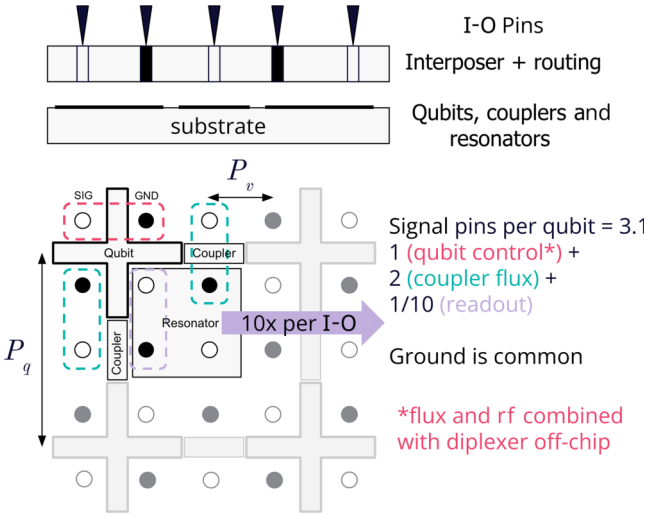

FIG. 4. A schematic layout for the required components to couple superconducting qubits through tunable couplers in a surface-code layout. In this example, we demonstrate the essential components: qubits, readout resonators, tunable couplers, and I-O pins. Each physical qubit and coupler requires a single control line, their respective ground connections, and an additional rf line for multiplexed readout shared among ten qubits. This limits the physical-qubit pitch to match that of the signal lines. A qubit-to-qubit pitch at *three* times the signal-line pitch has sufficient I-O pins to accommodate this.

each signal must interface with the wiring and, eventually, with the control electronics. Any significant fan-out near the physical-qubit chip would be costly; therefore, we assume a compliant joint to establish a reusable interconnect with a printed circuit board. With this assumption, a $P_v = 333$ μm interposer would result in a $P_q = 1$ mm. This leads to a physical qubit density of 10^6 qubits per m^2 assumed in our architectural model.

Fabricating a single device containing hundreds of thousands or even millions of physical qubits may seem unrealistic by today's standards, but we can assess where the fundamental limitations lie. Most features in a superconducting-qubit circuit extend beyond the micron scale, where existing processes for flat-panel displays can easily handle critical dimensions. Consequently, the distributed and lumped-element microwave circuitry and TSVs could likely be produced on substrates at the m^2 scale. However, the most fundamental component of the superconducting transmon qubit is the Josephson junction (JJ) [56], which is often defined using electron-beam lithography [57]. In this context, yield and manufacturing precision are crucial for frequency targeting and other factors. While recent advances have improved the tunability of JJs, leading to more scalable manufacturing methods [58,59], adapting an appropriate JJ fabrication process at the m^2 scale may prove unattainable.

In contrast, tiling superconducting dies on a single monolithic carrier is feasible, necessitating the execution

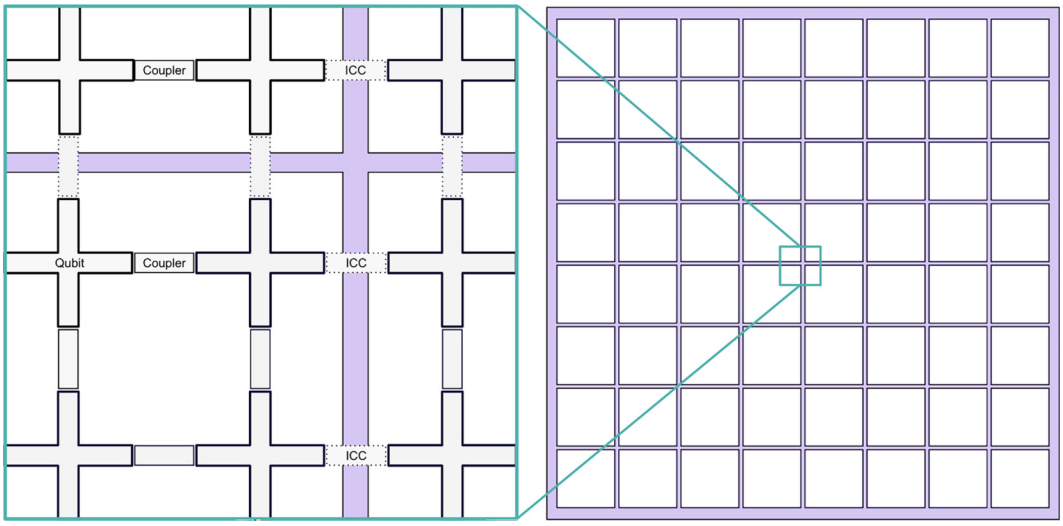

FIG. 5. Physical-qubit chips are tiled on a carrier. Tunable couplers manage two-qubit gates on physical qubits within the chip. ICC refers to "interchip coupler."

of high-fidelity two-qubit gates across the interface between adjacent physical-qubit dies. This concept has been demonstrated in Ref. [5], where a functionally similar tunable coupler [60], commonly used in high-performing superconducting transmon qubit systems, can bridge the gap between chips. Based on this, we develop our architectural model on physical-qubit chips tiled on a common carrier, featuring native fourfold connectivity across the entire carrier Fig. 5. While other yield concerns exist with multichip modules, as noted in Ref. [61], we assume that dies are prescreened and defect free in this paper.

1. Decoder and noise modeling

To conduct physically grounded device-specific resource estimates, we have developed an error-scaling pipeline that integrates the effects of decoding and the noise properties of superconducting systems. This pipeline converts a device-level noise model into the physical-to-logical error scaling law used in our resource counts. As part of our pipeline, we selected the square planar iSWAP surface code [21] as implemented by the MIDOUT package [62]. To generate input noise models for the QEC simulations, we create physical-device-inspired two-qubit correlated noise models for our iSWAP gates. We then apply the resulting terms in a Pauli error model to simulate the noise process in a standard Clifford tableau simulator, STIM [63]. For simplicity in this initial work flow, we use a minimum-weight perfect matching (MWPM) decoder implemented in the PyMatching software package [64].

We assume a power-law physical-to-logical error scaling law for a single cycle of the surface code [7,65]:

$$p_C = \kappa \left(\frac{p}{p_{\text{thresh}}} \right)^{(d+1)/2}, \tag{1}$$

where p_C is the per-cycle logical error rate and p denotes the homogeneous, mean, or dominant physical error rate. κ and p_{thresh} are fitting parameters. As part of our MWPM decoder simulations and device-noise modeling, we sweep through d values for one- and two-qubit error channels, which leads to $p_C[\text{MWPM}] \approx 0.009\,(p/0.016)^{(d+1)/2}$, i.e., $\kappa[\text{MWPM}] = 0.009$ and $p_{\text{thresh}}[\text{MWPM}] = 0.016$. The scaling coefficients, although consistent with familiar values for surface codes, are illustrative and depend on the chosen simulation configuration. The resource estimates that follow are commensurate with this choice. Though simplistic, this choice of noise model and decoder has been made to demonstrate how RRE allows the user to specify the coefficients appropriate to their device and decoder—see Appendix D for additional noise modeling details leading to the coefficients above, as well as Sec. V for an alternative approach using distinct scaling coefficients.

2. Intramodule microarchitecture dimensions

We aim to estimate the overall resources required to execute the algorithm on the FTQC depicted in Fig. 3, comprising two sets of modules. To this end, we present estimates for the spatial dimensions of the intramodule microarchitecture, which is identical across all modules and used for all resource estimates, illustrated in Fig. 1.

Each module must manage graph-state preparation, consumption, local MSD, T, and Bell-state routing. Within each module, the FTQC executes the algorithm based on the ASG formalism through intra- and intermodule operations on a wrapped logical memory register (blue tiles), mediated by continuous combs of logical auxiliary qubits (yellow and light-yellow tiles in Fig. 1). The blue quantum memory chain is part of the complete memory register

shared among all $n_{\text{modules}}^{\text{per-leg}}$ modules on the same leg of the macroarchitecture. At any moment, auxiliary bus portions on the same leg can communicate through d-wide coherent interconnects. The yellow auxiliary bus mediates logical measurements from the schedules on most of the blue logical memory qubits. In contrast, the light-yellow auxiliary+ bus performs the same task on the otherwise inaccessible corner memory tiles and routes the Bell pairs required for the graph handover. We assign two connected auxiliary and supplementary buses to simplify resource-scaling calculations, maintain symmetry in intramodule patterns, and highlight their roles; these auxiliary bus elements are otherwise identical. More efficient configurations may exist for embedding intramodule components, depending on problem requirements—Fig. 1 has been selected solely for its simplicity and scalability.

The quantum memory register across all modules on the same leg has a logical size of $\tilde{n}_{\text{logical}}$, totaling $2\tilde{n}_{\text{logical}}$ patches for the macroarchitecture. Modules have a designated area comprising at least $2\lceil \tilde{n}_{\text{logical}}/n_{\text{modules}}^{\text{per-leg}} \rceil$ patches for the blue memory register and yellow auxiliary bus portions. We allocate one complete column and one connected tile to the auxiliary+ bus ($l_{\text{edge}}+1$ patches on the far left side in Fig. 1), then assign the memory register and auxiliary bus portions to one complete edge of an individual module. The memory register and auxiliary bus portions form a symmetric bilinear comb pattern repeated throughout the modules. The bilinear pattern occupies as much space as necessary to support graph-state processing on $\lceil \tilde{n}_{\text{logical}}/n_{\text{modules}}^{\text{per-leg}} \rceil$ memory nodes following the schedules. The last module structures, which may require fewer active patches for the memory register and auxiliary bus portions, remain unchanged for consistency.

The FTQC performs MPPOs of $\tilde{S}_{\text{prep}}^{\mathcal{G}}$ and T-basis measurements of $\tilde{S}_{\text{consump}}^{\mathcal{G}}$ on the blue logical memory qubits via the yellow and light-yellow bus patches. Therefore, the buses must be connected and continuous, and they must have clear access to memory patches as required by the schedules. The bilinear memory register and auxiliary bus pattern must be at least *three* tiles wide to support its comblike pattern, enabling complete graph-state processing. The logical width of the bilinear pattern per module (in the direction perpendicular to the boundaries of the bilinear pattern and the T-transfer bus) is given by

$$l_{\text{qbus}} = \max\left(\left\lceil \frac{\left\lceil \frac{\tilde{n}_{\text{logical}}}{n_{\text{per-leg}}^{\text{modules}}} \right\rceil}{2\left\lfloor \frac{l_{\text{edge}}-2}{4} \right\rfloor + 1} \right\rceil, 3\right). \tag{2}$$

The unallocated space, the blank patches in Fig. 1, is not explicitly assigned but can still serve as additional buses for $O(1)$ operations.

Depending on the required accuracy for the complete FT algorithm and other architectural configurations, we select $n_{T\text{ factories}}$ copies of a suitable T factory (also known as a T distillery, magic state distillery, or distillation widget) per module for MSD, which generates purified T states at all steps as needed for consumption operations (see also Refs. [9,28] and Sec. II D 7). We place the T factories on the opposite side of the bilinear pattern. The number of T factories per module is

$$n_{T\text{ factories}} = \left\lfloor \frac{l_{\text{edge}}-1}{\lceil \frac{L_{\text{width}}}{\sqrt{2}d} \rceil} \right\rfloor n_{T\text{ factories}}^{\text{col}} \geq 1, \tag{3}$$

and each has a physical size $L_{\text{length}} \times L_{\text{width}}$. Furthermore, the number of T factory columns (purple in Fig. 1) in each module is given by

$$n_{T\text{ factories}}^{\text{col}} = \left\lfloor \frac{l_{\text{edge}}-l_{\text{qbus}}-1}{\lceil \frac{L_{\text{length}}}{\sqrt{2}d} \rceil + 1} \right\rfloor. \tag{4}$$

The quantum memory and auxiliary portions also serve as the endpoint for graph consumption. T-state teleportations are *not* permitted between modules, whether they are on the same or different legs. This means that all T-state distillation, movements (as exemplified by arrows in Fig. 1), and consumption occur locally within a module. During a graph-consumption step, the local generation, queue, and transfer of T states continue to support all T- and R_z-basis measurements on graph-state nodes corresponding to the subschedule operations. We have introduced a specific midway auxiliary system, called a T-transfer bus or buffer (teal in Fig. 1), to streamline this process. Once we allocate the quantum memory and auxiliary portions, we can arrange the T-transfer bus in another comblike pattern wrapped around T factories to provide a continuous interface and maximize $n_{T\text{ factories}}$. The T-transfer bus is responsible for actively queuing the magic states distilled by $n_{T\text{ factories}}$ T factories, up to its logical length, given by

$$l_{\text{transfer-bus}} = \left(l_{\text{edge}} - l_{\text{qbus}} - n_{T\text{ factories}}^{\text{col}} \left\lceil \frac{L_{\text{length}}}{\sqrt{2}d} \right\rceil\right) l_{\text{edge}} + n_{T\text{ factories}}^{\text{col}} \left\lceil \frac{L_{\text{length}}}{\sqrt{2}d} \right\rceil. \tag{5}$$

The T-transfer bus includes a central buffer area depicted in Fig. 1. This buffer connects to the auxiliary qubits in the bilinear pattern along one entire edge of the module. In the simplest form of T routing for graph consumption, the architecture ensures that the T-transfer bus can concurrently service at most $n' = \min(n_{T\text{ factories}}, n_{\text{qbus}}^{\text{row}})$ nodes of the auxiliary bus, which is a conservative lower limit on the number of simultaneous T states accessible. The FTQC moves the queued distilled T states to the

required patch of the auxiliary buses, strictly local to the module, via lattice-surgery-based patch-deformation operations [10,24], which cost $O(1)$ in the unit of the standard period of d code cycles, sometimes referred to as a "tock" [10]. In our FT resource estimates, we assume that these operations incur no additional temporal costs, i.e., they can be performed in parallel with other logical operations or with negligible delays. Here, $n_{\text{qbus}}^{\text{row}} = \left\lfloor \left(\left\lceil \tilde{n}_{\text{logical}} / n_{\text{per-leg}}^{\text{modules}} \right\rceil + 1 \right) / l_{\text{qbus}} \right\rfloor$. Note that there can be at most $\left\lceil \tilde{n}_{\text{logical}} / n_{\text{per-leg}}^{\text{modules}} \right\rceil$ nodes requiring T-basis measurements simultaneously at any given time. Upon allocating all essential components, there may be none or multiple partially or fully unallocated columns of logical qubits left on the right side; Fig. 1 shows a single leftover column as an example.

3. Computation of thermal resources of hardware

In addition to space-time volume data, including algorithmic execution times, physical space, and logical qubit counts, RRE can identify the hardware resources required to execute specific algorithms. These include counts of signal lines, the functional chip area, and the number of amplifiers, among others. Heat loads are a significant concern for superconducting qubits operating at millikelvin temperatures. Thus, we also estimate the power consumption, dissipation, and energy needed to run the FTQC.

Cryogenic energy-consumption calculations for the proposed architecture are based on per-line thermal loads. These loads are calculated assuming an all-superconducting signaling solution with input and output lines below 4 K, conventional dispersive readout using a traveling-wave parametric amplifier (TWPA), and high-electron-mobility transistors (HEMTs). We assume that each physical qubit requires a microwave line for XY control and a flux bias line for tuning the qubit frequency. Each pair of physical qubits includes a tunable coupler [66] that requires a single flux line, as shown in Fig. 4. Dispersive readout is expected to read out ten physical qubits multiplexed in parallel. Heat loads are currently dominated by HEMT dissipation and static heat loads, so only those are considered. Example numbers are provided in RRE based on estimates from Ref. [67].

The power consumed by the overall system is sensitive to the efficiency of cooling the FTQC. We scale overall power consumption using rough estimates for the power required to operate 4-K cryoplants (single efficiency factor of 500 kW of power for 500 W of cooling at 4 K) and dilution refrigerators (single efficiency factor of 1 kW of power for 1 μW of cooling at 20 mK), as outlined in the published work [68] and manufacturer specifications—see also parameter 49 of Appendix A. We anticipate that these estimates will evolve, as they depend on the detailed design of the signal chain.

D. Modular graph-state-based compilation and resource estimations

We assume that the user has a fixed description of their logical and hardware architecture, represented by a set of configuration parameters, denoted by {config}. The user also provides a circuit that implements a unitary operator U acting on n_{input} logical qubits of the FT algorithm:

$$|\text{output}\rangle = U\,|\text{input} := s_0 = 0, s_1 = 0, \ldots, s_{n_{\text{input}}-1} = 0\rangle\,. \tag{6}$$

In practice, U consists of a sequence of quantum instructions that may include tens of millions of logical gates. We always start with an all-zero $|\text{input}\rangle$ state to simplify calculations.

One can always transpile the initial circuit to express it solely in terms of logical Cliffords, T, $T^\dagger$, and arbitrary-angle non-Clifford R_Z gates, i.e., $U\left(\{\text{Clifford}, T, T^\dagger, R_z(\theta)\}\right)$. We denote the exact number of original logical T and $T^\dagger$ gates, excluding those implicitly included within the arbitrary-angle rotations, as n_{init}^{T}. Moreover, $n_{\text{init}}^{R_z}$ represents the exact number of arbitrary-angle non-Clifford R_Z gates in U, including those implicitly in the logical gates (excluding T and $T^\dagger$ operations counted separately in n_{init}^{T}). We assume FT execution of $U\left(\{\text{Clifford}, T, T^\dagger, R_z(\theta)\}\right)$ using rotated surface-code patches, as described in Refs. [10,22,23], with a code distance d. In our approach, the decomposition of non-Clifford rotations takes place after compilation during the consumption stages of the graphs (see Fig. 2). This process is sometimes referred to as gate synthesis [69,70]. We denote the diamond-norm precision for gate syntheses as ε.

We consider an MBQC framework that uses ASG compilation for circuit execution. One advantage of this approach is that it does not require initializing the entire graph state, $\mathcal{G}$, on FTQC at all times [33,71]. Instead, we split circuit execution into multiple time steps, each containing a subcircuit or widget with its own (sub)graph. We identify the parts of the graph state needed at any moment by constructing three complete schedules by sequentially adding subschedules across time steps: the preparation schedule for the widgetized algorithm, labeled as $\tilde{S}_{\text{prep}}^{\mathcal{G}}$. The consumption schedule for the widgetized algorithm will be labeled as $\tilde{S}_{\text{consump}}^{\mathcal{G}}$. Additionally, a related schedule specifies the measurement bases used by FTQC during the consumption stage, which we refer to as $\tilde{S}_{\text{meas}}^{\mathcal{G}}$.

We verify that the widgetized schedules $\{\tilde{S}_{\text{prep}}^{\mathcal{G}}, \tilde{S}_{\text{consump}}^{\mathcal{G}}, \tilde{S}_{\text{meas}}^{\mathcal{G}}\}$ provide a faithful representation of the preparation and consumption of the quantum state $\mathcal{G}$, in accordance with U. This is equivalent to any valid set of $\{S_{\text{prep}}^{\mathcal{G}}, S_{\text{consump}}^{\mathcal{G}}, S_{\text{meas}}^{\mathcal{G}}\}$ when time-sliced widgetization is not used. Previous studies [10,32–34,44–46] have shown

that valid arrangements of such schedules can effectively prepare [34] and consume [33] the graph, highlighting their connection to its spatial and temporal structure. Our focus will be on introducing the notation for $\{\tilde{S}^{\mathcal{G}}_{\text{prep}}, \tilde{S}^{\mathcal{G}}_{\text{consump}}, \tilde{S}^{\mathcal{G}}_{\text{meas}}\}$.

$\tilde{S}^{\mathcal{G}}_{\text{prep}}$ is a time-ordered list of n_{widgets} preparation subschedules, each corresponding to a widget. The FTQC must follow these subschedules to initialize subgraphs in several substeps using n-body CZ and single-qubit Cliffords [32,34,44] that correspond to lattice surgery MPPOs for X and Z stabilizers [10,24–27]. Specifically, $\tilde{S}^{\mathcal{G}}_{\text{prep}} = \{\tilde{S}^{g_0}_{\text{prep}}, \ldots, \tilde{S}^{g_{n_{\text{widgets}}-1}}_{\text{prep}}\}$. Each preparation subschedule may comprise multiple substeps, represented as $\tilde{S}^{g_i}_{\text{prep}} = \{\tilde{S}^{g_i^0}_{\text{prep}}, \ldots, \tilde{S}^{g_i^{J(i)}}_{\text{prep}}\} \forall\, i \in \{0, \ldots, n_{\text{widgets}} - 1\}$. Furthermore, a $\tilde{S}^{g_i^j}_{\text{prep}}$ can contain several nonoverlapping MPPOs [34,44]. The MPPOs within a substep can be executed simultaneously using the auxiliary buses.

Similarly, $\tilde{S}^{\mathcal{G}}_{\text{consump}}$ is a sequence of n_{widgets} sets, with each set serving as the consumption subschedule for a corresponding widget. To implement the original non-Cliffords, the FTQC must follow these subschedules, performing logical arbitrary-angle $R_Z(\theta)$-basis measurements on the relevant nodes in multiple substeps. In detail, $\tilde{S}^{\mathcal{G}}_{\text{consump}} = \{\tilde{S}^{g_0}_{\text{consump}}, \ldots, \tilde{S}^{g_{n_{\text{widgets}}-1}}_{\text{consump}}\}$. Each of these consumption subschedules may contain several substeps, represented as $\tilde{S}^{g_i}_{\text{consump}} = \{\tilde{S}^{g_i^0}_{\text{consump}}, \ldots, \tilde{S}^{g_i^{K(i)}}_{\text{consump}}\} \;\forall\; i \in \{0, \ldots, n_{\text{widgets}} - 1\}$. A specific $\tilde{S}^{g_i^k}_{\text{consump}}$ can contain multiple nonoverlapping measurement lists, as detailed in [33,46]. Measurements in nonoverlapping lists within a substep can be executed simultaneously. Meanwhile, $\tilde{S}^{G}_{\text{meas}}$ is a connected sequence that specifies the rotational basis (T or $R_Z(\theta)$) required for node measurements across the corresponding substeps of $\tilde{S}^{\mathcal{G}}_{\text{consump}}$. This method distinguishes two categories of node measurements during the consumption stage: those that require T-basis measurements, consuming a single distilled T state, and those that require $R_Z(\theta)$-basis measurements, consuming multiple distilled T states following gate synthesis of $R_Z(\theta)$ to a Clifford+T sequence.

The number of nodes in $\mathcal{G}$, called the *graph size* and denoted by N, is typically much larger than the number of input logical qubits, $N \gg n_{\text{input}}$. This is because we always add logical auxiliary qubits to implement an MBQC of a given quantum algorithm. Moreover, N is always greater than n_{input} [32,46,71]. However, due to the scheduling requirements above, only a relatively small subset of logical qubits, each representing a graph node, is required at any given step to execute the unitary operation U. We denote the maximum number of logical qubits needed at any step as $\tilde{n}^{\mathcal{G}}_{\text{logical}}$, i.e., the maximum quantum memory required. Typically, $n_{\text{input}} < \tilde{n}^{\mathcal{G}}_{\text{logical}} < N$ [33,46].

One can create distinct graph states with similar attributes from different input states, but the same U. We construct $\mathcal{G}$ by *stitching* graphs from each time step for resource estimations. To admit arbitrary input nodes into these ASGs, we always initialize all nodes in the all-zero input state, $|0, 0, \ldots, 0\rangle$, and then replace the input nodes with n_{input} copies of $(|0+\rangle + |1-\rangle)^{\otimes n_{\text{input}}}$, attaching the appropriate auxiliary nodes. This process allows us to teleport from the output nodes of one (sub)graph to the input nodes of another in the consecutive time step [32,46], which we use to verify and stitch graph states.

To summarize, we can achieve *exact* resource estimates for U by providing an MBQC-based set of primitives for the nonwidgetized algorithm:

$$S_{\text{est}}(U, \{\text{config}\}) := \{d, \varepsilon, n^{T}_{\text{init}}, n^{R_z}_{\text{init}}, n^{\mathcal{G}}_{\text{logical}}, S^{\mathcal{G}}_{\text{prep}}, S^{\mathcal{G}}_{\text{consump}}, S^{\mathcal{G}}_{\text{meas}}\}. \quad (7)$$

In this equation, the required user inputs are explicitly specified on the left-hand side, whereas the right-hand side yields the corresponding set of primitives for resource estimations. It becomes extremely challenging to construct S_{est} for a practical U containing tens of millions of gates and considerable circuit depth, whether for resource estimation or hardware execution. Therefore, we proposed a time-sliced widgetization of U that yields an approximate resource-estimation set $\tilde{S}_{\text{est}}$. Below, we will provide a step-by-step recipe for creating and rigorously verifying $\tilde{S}_{\text{est}}$.

1. RRE *software work flow*

Our FT resource-estimation tool, RRE [29], has been extensively developed and used to generate the results presented in this work. The RRE estimation strategy follows a work flow that translates a logical circuit into an ASG, maps it onto the logical layout shown in Fig. 1, and then assesses the required physical resources. This section provides a brief overview of the RRE step-by-step work flow, while detailed explanations of selected steps appear in subsequent sections:

(1) *Widgetized logical circuit and other parameter inputs*. The user provides an FT algorithm, U, as a custom JSON file, which can contain a single OpenQASM 2.0 circuit or be widgetized into multiple OpenQASM 2.0 subcircuits, crucial for large-scale applications. For the widgetization strategy that we have employed, see Sec. II D 2. To estimate resources, in addition to supplying a logical circuit, the user must also set a set of system architectural parameters, {config}, including physical error rate, p_{logical}-to-p scaling coefficients (based on decoder and device models), cycle times, and the desired total failure-rate budget, $p_{\text{algo-fail}}$. RRE users can select preconfigured {config} and modify them through an input YAML file.

(2) *Transpilation*. RRE parses the OpenQASM 2.0 circuit(s) and counts the initial T gates, n_{init}^{T}, and all arbitrary-angle rotation gates, $n_{\text{init}}^{R_z}$, in the FT algorithm, as needed for later steps. It replaces any unsupported unitary in the circuit(s) with *exact* gate equivalents when the downstream fault-tolerant compiler does not support it (for details, see Sec. II D 4).

(3) *Graph-state processing and compilation*. The CABALISER software (see Sec. II D 5 and Ref. [46]) is used to compile the transpiled circuits into ASGs. The graph objects are cached and reusable. Even if the architecture changes, the portable graph objects can be used to estimate resource requirements for that algorithm without reprocessing the graph states.

(4) *Substrate scheduling*. The `Substrate Scheduler` is called to generate preparation schedules for ASGs, as detailed in Sec. II D 6 and Ref. [44].

(5) *Find the required precision and distance iteratively*. Given $p_{\text{algo-fail}}$, the graphs, and the preparation schedules, iterate through a look-up widget table to find ε, d, and the optimal T factory (with output logical error rate p_{out}) that satisfies $\varepsilon < p_{\text{logical}}$ and $p_{\text{out}} < p_{\text{logical}}$. ε determines the length of the Clifford+T chain, L_ε, for arbitrary angle transformations needed at the final measurement points of the consumption schedules.

(6) *Identify space complexity*. RRE determines the total physical-qubit count to be allocated to the hardware, n_{physical}, which scales with $\tilde{n}_{\text{logical}}$, the maximum node degree of the graph, our proxy for the number of logical qubits required. In practice, $\tilde{n}_{\text{logical}}$ will only provide an upper bound on the required logical qubit count.

(7) *Identify time complexity*. Given d and the characteristic timescales set in the architectural configurations of the system, config, find $\{t_{\text{tock}}^{\text{quantum}}, t_{\text{tock}}^{\text{decoding}}, t_{\text{tock}}^{\text{intermodule}}\}$.

(8) *Find full `physHW` object and hardware times*. RRE uses the tocks, schedules, and other architectural configurations from config (cryostat sizes, line, coupler, and power-dissipation details) to identify an FT hardware object, `physHW`, including the number of couplers, modules, area, memory, energy usage, etc. `physHW` fully identifies all outputs including total FT hardware time, decoding core numbers, and total power and energy consumption. Hardware execution times are computed from MPPOs across auxiliary buses for local and intermodule interactions.

(9) *Display results*. Tabulate and print resource-estimation outputs to the console and a `CSV` file as requested. All 48 default RRE output parameters are detailed in Appendix A.

2. Input-circuit widgetization

Due to the large scales involved in a practical input circuit, a decomposition strategy is essential to avoid memory bottlenecks on RRE. Luckily, quantum algorithms usually consist of repeating blocks, making their decomposition easier.

Our decomposition strategy slices the unitary operator U along the time direction into n_{widgets} widgets (subcircuits) of fixed width n_{input}. Each widget may have a distinct emergent depth. Crucially, no entangling gates connect adjacent subcircuits. We can then execute the complete algorithm as a sequence of these subcircuits, expressed as $U = U_{n_{\text{widgets}}-1} \cdots U_1 U_0$, where

$$U_i \,|\text{state}\rangle_i = |\text{state}\rangle_{i+1} \quad \text{for } i \in \{0, \ldots, n_{\text{widgets}} - 1\}. \tag{8}$$

Here, $|\text{state}\rangle_0 \equiv |\text{input}\rangle$ and $|\text{state}\rangle_{n_{\text{widgets}}} \equiv |\text{output}\rangle$.

In a typical quantum algorithm, many widgets may be equivalent and repeated along the time direction. In such cases, we need to compile only the set of distinct widgets, e.g., $[U_0, U_4, \ldots, U_{n_{\text{widgets}}-7}]$. We use n_{widgets}^{*} to represent the number of distinct widgets in the set $[U_0, U_1, \ldots, U_{n_{\text{widgets}}-1}]$ ($n_{\text{widgets}}^{*} \leq n_{\text{widgets}}$). We refer to this process as *circuit widgetization*, illustrated in Fig. 6. We base our choice of widgetization strategy on the creation of subcircuit dependency graphs. This approach allows us to compile large circuits into elementary operations and unitarily recombine compiled widgets, avoiding

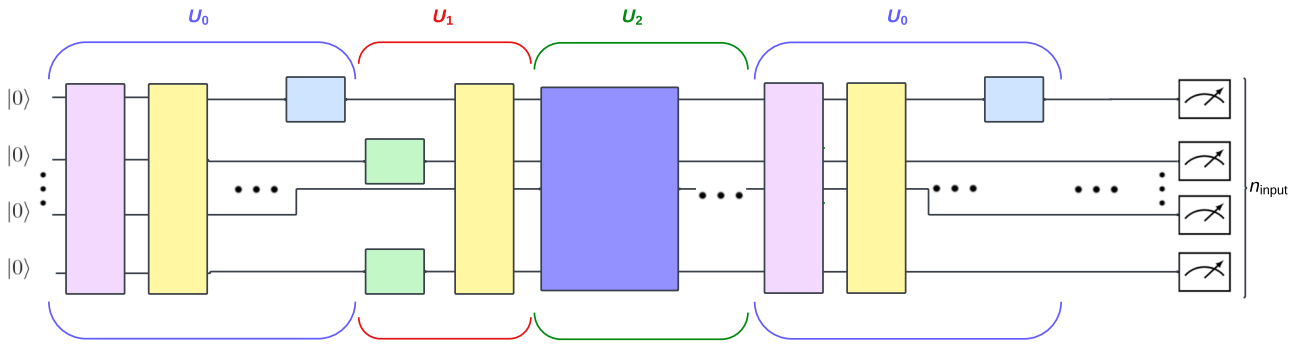

FIG. 6. The input-circuit decomposition strategy. This visualization demonstrates how we "widgetize" a complete logical algorithm, U, into n_{widgets} time-sliced widgets, represented as $U_{n_{\text{widgets}}-1} \cdots U_1 U_0 \equiv U$. Each widget has the same width, n_{input}, but varying depth. Some widgets may be repeated in the time direction (e.g., $U_3 \equiv U_0$ above), allowing us to compile only n_{widgets}^{*} distinct subcircuits.

excessive memory overhead. Compilation is essential for executing the algorithm and for accurate resource estimates. Additionally, this strategy helps keep the number of distinct widgets n^*_{widgets} relatively low, facilitates implementation, and justifies using time slicing with a fixed n_{input}. However, this particular widgetization choice is likely not optimized to minimize space-time costs. We present further details on our widgetization strategy in Appendix B.

3. Stitching strategy

Following the widgetization process, we can construct an approximate version of S_{est}. We propose the widgetization-based set $\tilde{S}_{\text{est}}$ as a good approximation of S_{est} for resource estimations, defined as follows:

$$\begin{aligned}\tilde{S}_{\text{est}} &:= \{d, \varepsilon, n^T_{\text{init}}, n^{R_z}_{\text{init}}, \tilde{n}^{\mathcal{G}}_{\text{logical}}, \tilde{S}^{\mathcal{G}}_{\text{prep}}, \tilde{S}^{\mathcal{G}}_{\text{consump}}, \tilde{S}^{\mathcal{G}}_{\text{meas}}\} \\ &\approx S_{\text{est}}. \end{aligned} \tag{9}$$

Recall that n^T_{init} and $n^{R_Z}_{\text{init}}$ are the exact counts of explicit $\{T, T^\dagger\}$ and all arbitrary-angle R_Z operations in U. In this widgetization-based estimation procedure, we first obtain estimation primitives for each widget based on its (sub)graph. We then obtain the overall circuit resource estimates as follows:

$$\begin{aligned}\tilde{n}^{\mathcal{G}}_{\text{logical}} &:= \max_{g_i}(n^{g_i}_{\text{logical}}) \approx n^{\mathcal{G}}_{\text{logical}}, \\ \tilde{S}^{\mathcal{G}}_{\text{prep}} &:= \cup_{i=0}^{n_{\text{widgets}}-1} S^{g_i}_{\text{prep}} \equiv S^{\mathcal{G}}_{\text{prep}}, \\ \tilde{S}^{\mathcal{G}}_{\text{consump}} &:= \cup_{i=0}^{n_{\text{widgets}}-1} S^{g_i}_{\text{consump}} \equiv S^{\mathcal{G}}_{\text{consump}}, \\ \tilde{S}^{\mathcal{G}}_{\text{meas}} &:= \cup_{i=0}^{n_{\text{widgets}}-1} S^{g_i}_{\text{meas}} \equiv S^{\mathcal{G}}_{\text{meas}}. \end{aligned} \tag{10}$$

Here, g_i represents the graph state compiled from the ith widget.

Equation (10) formalizes our stitching strategy and outlines the key assumptions underlying our estimates. It identifies how we combine the individual graphs and scheduling lists to approximate the complete $\mathcal{G}$, as well as $S^{\mathcal{G}}_{\text{prep}}$, $S^{\mathcal{G}}_{\text{consump}}$, and $S^{\mathcal{G}}_{\text{meas}}$. While an FTQC must physically stitch the widgets (through teleportations) to execute the complete circuit, this is unnecessary for resource estimation. Consequently, we have reduced the computational cost of obtaining resource estimates by computing the widgetized quantities in Eq. (10) through software.

4. Preprocessing the input subcircuits

To prepare for graph compilation, we must parse each widget and *transpile* it to include only logical Clifford + $T + R_Z(\theta)$ gates, using exact gate equivalences. Here, the θ values represent non-Clifford angles. Later, during the consumption stage, when we want to perform single-qubit logical measurements, we need to further decompose each $R_Z(\theta)$ node into a sequence of Clifford+T gates through gate synthesis [69,70].

We perform transpilation to the logical Clifford + T + $R_Z(\theta)$ using RRE. Performing gate synthesis at the end of the consumption stage enables us to generate, cache, and reuse the set of subgraphs $\{g_0, \ldots, g_{n_{\text{widgets}}-1}\}$ and their attributes independently of the logical and hardware-level configurations, {config}.

Once transpilation is complete, the next step is to generate the unified MBQC-based set of graph states $\{g\}$ and schedules $\{S^g_{\text{prep}}\}$, $\{S^g_{\text{consump}}\}$, and $\{S^g_{\text{meas}}\}$, and other essential intermediary lists detailed below, which yields $\tilde{S}_{\text{est}}$ sufficient for resource estimation of U.

5. CABALISER*: Compiling to graphs and consumption schedules*

RRE uses the stabilizer tableau-based FT compiler CABALISER [46] as a crucial intermediate layer to transform the set of individual logical widgets, $\{U\}$, into corresponding MBQC-based subgraphs, consumption schedules, "locally simulated" quantum states, and other essential information. The outputs of CABALISER are locally simulated quantum states derived from exact calculations. CABALISER effectively maps an input state $|\text{state}\rangle_{i-1}$ to $|\text{state}\rangle_i^{\text{CABALISER}}$: it applies U_i, Pauli corrections, the consumption subschedule, and additional logical operations (such as Hadamards) to the $(i-1)$th graph state to create a faithful vector representation of the corresponding universal subgraph at step i. It is guaranteed that $|\text{output}\rangle \equiv |\text{state}\rangle^{\text{CABALISER}}_{n_{\text{widget}}-1}$. This vector representation helps verify the unitariteness of RRE and of the CABALISER outputs for small n_{input}. We present a systematic method for verifying the graphs and schedules produced by RRE, accounting for widgetization and stitching in Appendix C. For complete details on scalability and various compilation components of CABALISER, see Refs. [32,38,46].

For a given individual subcircuit U_i that includes only Clifford + T + $R_Z(\theta)$, CABALISER can generate the following set of quantities:

$$\begin{aligned} U_i(\{\text{Clifford}, T, T^\dagger, R_z(\theta)\}) &\xrightarrow{\text{CABALISER}} \\ \{g_i, S^{g_i}_{\text{frames}}, S^{g_i}_{\text{input}}, S^{g_i}_{\text{output}}, n^{g_i}_{\text{logical}}, \\ S^{g_i}_{\text{consump}}, S^{g_i}_{\text{meas}}, |\text{state}\rangle_i^{\text{CABALISER}}\}. \end{aligned} \tag{11}$$

In this equation, $S^{g_i}_{\text{frames}}$ is a set that tracks the required Pauli corrections (frames) to be applied to measurements at the end. $S^{g_i}_{\text{output}}$ and $S^{g_i}_{\text{input}}$ represent the sets (or maps) of output and input nodes in the graph state, respectively. Because we decompose the entire algorithm in the time direction with a uniform width that covers all logical qubits, the number of intermediate input and output nodes remains the same. $n^{g_i}_{\text{logical}}$ denotes the quantum memory needed to process U_i. $S^{g_i}_{\text{consump}}$ identifies the graph consumption

schedule for performing individual logical qubit measurements. $S^{g_i}_{\text{meas}}$ identifies which rotational basis to use for each measurement during the consumption stage. Finally, $|\text{state}\rangle_i^{\text{CABALISER}}$ represents the locally simulated quantum state extracted from the outputs generated up to the ith step.

We repeat the process of Eq. (11) for all n^*_{widgets} distinct widgets, providing the user with the information required for the approximate estimation set in Eq. (9), excluding the preparation schedules, $\{S^g_{\text{prep}}\}$, which we generate using another tool detailed below.

6. Substrate Scheduler: Generating preparation schedules

RRE uses the Substrate Scheduler [44] to map the graph states $\{g\}$, computed by Cabiliser in Eq. (11), to a set of graph-preparation subschedules $\{S^g_{\text{prep}}\}$. These preparation subschedules are optimized for the logical memory register and auxiliary bus in the superconducting hardware layout shown in Fig. 1.

For the Substrate Scheduler calculations, we assume a dual-rail or bilinear quantum memory and auxiliary bus pattern consisting of linear arrangements of rotated surface-code logical qubits for storing graphs and mediating logical measurements on the nodes, as shown in Fig. 1. The pattern has a fixed size $\tilde{n}_{\text{logical}} := \max_{g_i}(n^{g_i}_{\text{logical}})$ per rail. This assumption ensures no memory constraints when executing the preparation and consumption stages consecutively. Although the intramodule microarchitecture used for resource estimates follows the layout shown in Fig. 1, the Substrate Scheduler results can still be matched and used under the assumption of a wrapped-bilinear pattern. Currently, the Substrate Scheduler identifies an optimal schedule based on the individual mapping of each g to this layout. The performance of the Substrate Scheduler could be enhanced by incorporating additional information about node types and supporting customized bus architectures, which we plan to address in future work.

7. Space-time-optimal resource estimation

In this section, we formulate the space-time volume requirements for graph-state processing parts based on {config}, $\tilde{S}_{\text{est}}$, and the architectures presented in Sec. II C, with a spatially optimal size of the quantum register and time-optimal compilation strategies (given the spatial configurations), creating a "middle-of-the-way" design.

At a macroscopic level, the architecture features a ladder-style structure with $n^{\text{modules}}_{\text{per-leg}}$ pairs of QPU modules connecting the legs, as illustrated in Fig. 3. We have based this design on the assumption that superconducting platforms have a low syndrome extraction cycle and would benefit from minimizing the physical footprint while interleaving graph preparation and consumption. Consequently, we always create $\tilde{n}_{\text{logical}}$ rotated surface-code patches for the logical memory register shared among all modules on a leg of the macroarchitecture.

At the intramodule level, we organize each module as a square grid of logical qubits (patches), the foundational building blocks for all components. The edge size is determined by $l_{\text{edge}} = \left\lfloor \sqrt{n_{\text{phys}}/2d^2} \right\rfloor$, where $n_{\text{phys}} = 1 \times 10^6$ represents the total number of physical qubits available per module. For simplicity in resource estimates, we have assumed that each rotated surface-code patch contains $2d^2$ physical qubits.

We have performed a static allocation of intramodular components, providing sufficient resources for the largest widgets, as illustrated in Fig. 1. Each module has *five* main components, to which we allocate physical qubits. The unallocated or "wasted" space can still be used for ancillary operations. The first component is a quantum register of logical memory (graph-state) qubits. The second is a chain of logical auxiliary bus qubits of the same length, facilitating logical measurements on the memory qubits, and connects to the third component, i.e., the supplementary bus. The first two components are arranged in a snakelike bilinear pattern to fully occupy one QPU edge. The fourth component is the linear T-transfer bus, which supplies the auxiliary bus portion with T states for graph consumption. The last components are T factories (for details, see Sec. II C 2).

The space-time volume of the complete graph creation on the bilinear pattern before single-qubit logical measurements is

$$V_{\mathcal{G}} = 2\tilde{n}_{\text{logical}}\mathbb{L}_{\text{prep}}d, \tag{12}$$

where $\mathbb{L}_{\text{prep}}(\tilde{S}_{\text{prep}}) = \sum_{\text{sub}\in\tilde{S}_{\text{prep}}} L_{\text{sub}}$. Here, L_{sub} is the number of distinct measurement steps in the preparation subschedule of sub. In other words, $\mathbb{L}_{\text{prep}}$ is the total number of distinct measurement steps required for the complete graph preparation.

We use the same T factory, matched from an extended list of appropriate widgets, across all steps and modules. We build the extended T factory list using a method proposed by Litinski (see the original table 1 in Ref. [9]). We consistently assume a homogeneous superconducting physical error rate of $p = 10^{-3}$ for all quantum operations, which may originate from dominant error sources—whether from measurements or single- or two-qubit physical gates. We have sourced the matching widgets from numerical simulations in Ref. [9] and simulated a larger T factory, summarizing our findings in Table I as an extended "T factory look-up table." We recursively search this table to identify a suitable T factory. Notably, Table I is part of the user-provided configuration set, {config}, and can be customized in RRE.

In the "T factory look-up table" (see Table I), the output probability, p_{out}, determines the logical error rate for

TABLE I. T factory look-up table: Litinski's T factories that match our superconducting architecture with $p = 10^{-3}$. Extended from Ref. [9, table 1].

T factory (superconducting)	p_{out}	Physical space ($L_{\text{width}} \times L_{\text{length}}$)	Physical qubits, Q	Cycles, C
$(15\text{-to-}1)_{17,7,7}$	4.5×10^{-8}	64×72	4620	42.6
$(15\text{-to-}1)^{6}_{15,5,5} \times (20\text{-to-}4)_{23,11,13}$	1.4×10^{-10}	387×155	43 300	130
$(15\text{-to-}1)^{4}_{13,5,5} \times (20\text{-to-}4)_{27,13,15}$	2.6×10^{-11}	382×142	46 800	157
$(15\text{-to-}1)^{6}_{11,5,5} \times (15\text{-to-}1)_{25,11,11}$	2.7×10^{-12}	279×117	30 700	82.5
$(15\text{-to-}1)^{6}_{13,5,5} \times (15\text{-to-}1)_{29,11,13}$	3.3×10^{-14}	292×138	39 100	97.5
$(15\text{-to-}1)^{6}_{17,7,7} \times (15\text{-to-}1)_{41,17,17}$	4.5×10^{-20}	426×181	73 400	128
$(15\text{-to-}1)^{8}_{23,9,9} \times (15\text{-to-}1)_{49,19,21}$	9.0×10^{-23}	696×234	133 842	157.5

the output state by T factories and sets an upper limit on the error rate for any T gate implemented using the auxiliary qubits. The specifications for the number of physical qubits, the dimensions of the two-dimensional (2D) patch, and the number of physical cycle sweeps depend on the construction of the factory. Our analysis has focused exclusively on selecting T factories that produce T states, thus excluding any entries related to controlled-controlled-Z (CCZ) from the original Ref. [9, table 1].

Next, we must include the resource cost of decomposing non-Cliffords into Clifford + T (gate synthesis) to consume graph-state nodes via single-qubit measurements. After completing a preparation subschedule, at each substep of the consumption schedule, we use one or more purified T resources to perform measurements in the desired basis. The FTQC must move T states from T factories to the auxiliary bus through patch deformations [10,24] at negligible time cost $O(1)$. We assume that graph nodes either require arbitrary-angle R_Z-basis measurements or a single T-basis measurement. Measurements involving R_Z-axis rotations are decomposed into sequences of T gates (and Cliffords), with a fixed worst-case length of $L_\varepsilon = \lceil c_0 \log(1/\varepsilon) + c_1 \rceil$. We determine this length using the gate-synthesis methods available to RRE. Here, ε refers to the diamond-norm error for decompositions into Clifford+T (for further details, see [69,70]).

Generating each distilled T state in the T factories requires C cycles, and we need the d-cycle tock of one further surface code to interact with the graph node and measure it in either the X or Z basis. In this work, we ignore the time cost of any operation that requires $O(1)$ tocks (idling, deforming surface-code patches, etc.), assuming that we can perform them in parallel with other operations during substeps at no additional cost or with negligible delay. Therefore, for each non-Clifford measurement of memory nodes, assuming an R_Z basis as an example, the volume increases as follows:

$$2\tilde{n}_{\text{logical}}\mathbb{L}_{\text{prep}}d \to 2\tilde{n}_{\text{logical}}\mathbb{L}_{\text{prep}}d + (2\tilde{n}_{\text{logical}} + n^{\text{modules}}_{\text{per-leg}} l_{\text{transfer-bus}})(C+d)L_\varepsilon. \tag{13}$$

the second term reflects the fact that both graph node consumption and T-state distillation occur over a space of $2\tilde{n}_{\text{logical}} + n^{\text{modules}}_{\text{per-leg}} l_{\text{transfer-bus}}$. All nodes with distinct measurement basis types, T and R_Z, must be measured sequentially.

If we assume the worst-case sequential execution, the upper bound on the full space-time volume becomes

$$V^{\text{sequential}}_{\text{total}} = 2\tilde{n}_{\text{logical}}\mathbb{L}_{\text{prep}}d + (2\tilde{n}_{\text{logical}} + l_{\text{transfer-bus}})(N^{\text{seq}}_{\text{consump}}d + N^{\text{seq}}_{\text{distill}}C). \tag{14}$$

We define the total number of sequential T states for graph consumption as $N^{\text{seq}}_{\text{consump}} = \lceil n^T_{\text{init}}/n' \rceil + L_\varepsilon \lceil n^{R_z}_{\text{init}}/n' \rceil$. Recall that we have assumed a simplified T-routing scheme that allows the auxiliary bus to serve up to n' simultaneous magic states. FTQC can only consume distinct T and R_Z nodes in parallel. The number of sequential T states to distill is $N^{\text{seq}}_{\text{distill}} = \lceil N^T_{\text{tot}}/n' \rceil$, where the total T-state measurements required are defined as $N^T_{\text{tot}} := n^{R_Z}_{\text{init}} L_\varepsilon + n^T_{\text{init}}$. The quantity n^T_{tot} is the proxy that we use for T count and typically becomes larger than $N_{\mathcal{G}}$. We do not include T factory space-time volumes, as their errors subsume into the output probability p_{out} of the distilled state [9, fig. 1]. If the select widget from Table I has a second (20-to-4) layer (the second and third widgets), then we must replace $n' \to 4n'$ in all space-time equations.

As detailed in Sec. II C 1, a single-cycle step of the surface code (with multiple physical operation layers) has a failure probability p_C that characterizes the relationship between the physical and logical error rates, as given in Eq. (1). Consequently, we can express the logical failure probability of a d-cycle tock as

$$p_{\text{logical}} = 1 - (1 - p_C)^d. \tag{15}$$

The failure rate for a total volume of these d-cycle units, denoted as $V^{\text{sequential}}_{\text{total}}$, must satisfy the following condition:

$$p_{\text{algo-fail}} > 1 - (1 - p_{\text{logical}})^{V^{\text{sequential}}_{\text{total}}},$$

where $p_{\text{algo-fail}}$ represents the overall algorithm failure rate or error budget for executing the complete circuit U. This budget encompasses the entire space-time volume determined by the user within the parameter set {config}.

Assuming $p_C \ll 1$, we can approximate $(1-p_C)^d \approx 1 - p_C d$, leading us to conclude that $p_{\text{log-cell}} \approx p_C d$. Consequently,

$$p_{\text{algo-fail}} \gtrsim 1 - (1 - p_C d)^{V_{\text{total}}^{\text{sequential}}}.$$

This equation highlights the relationship between the failure rates and the characteristic space-time volume for the complete QEC process.

Subtracting each side of Eq. (16) from 1 and taking the logarithm of both sides, we obtain

$$\begin{aligned}\log\left(1 - p_{\text{algo-fail}}\right) &\lesssim V_{\text{total}}^{\text{sequential}} \log\left(1 - p_C d\right) \\ &\lesssim -V_{\text{total}}^{\text{sequential}} p_C d. \end{aligned} \tag{16}$$

If we substitute $V_{\text{total}}^{\text{sequential}}$ from Eq. (14) into Eq. (16), the space-time logarithmic inequality to solve can be expressed as

$$\begin{aligned}\kappa d \left(\frac{p}{p_{\text{thresh}}}\right)^{(d+1)/2} &\left(2\tilde{n}_{\text{logical}} \mathbb{L}_{\text{prep}} d + \left(2\tilde{n}_{\text{logical}}\right.\right. \\ &\left.\left. + n_{\text{per-leg}}^{\text{modules}} l_{\text{transfer-bus}}\right)\left(N_{\text{consump}}^{\text{seq}} d + N_{\text{distill}}^{\text{seq}} C\right)\right) < -J_1, \end{aligned} \tag{17}$$

where $J_1 = \log(1 - p_{\text{algo-fail}})$. The minimum code distance required is the smallest integer d that satisfies Eq. (17). The only free variables are the cycle time of the T factory, i.e., C, and ε. We have previously extracted $\mathbb{L}_{\text{prep}}$ and $\tilde{n}_{\text{logical}}$ from the CABALISER and Substrate Scheduler.

To determine C and ε, we perform nested loops to iterate through widgets and then compute ε of the gate synthesis. The inner loop feeds back into the Clifford + T decomposition, where the failure rate of a d-cycle tock, p_{logical}, sets the approximation error as $\varepsilon < p_{\text{logical}}$. By selecting a specific widget, we fix C, which also determines the output error rate of the distilled state, p_{out}.

In a compiled quantum algorithm, we aim to ensure that the distilled T states have precisely the same error rates as any other gates in the compiled circuit. A Pauli initialization or measurement occupies a d-cycle unit in the space-time volume. Thus, its failure rate is governed by Eq. (15). This error rate must be dominant. Therefore, the error output, p_{out}, from the distillation circuit should be less than the failure rate of this cell. That is, we must ensure that the distillation output is at least as good as a unit cell in the space-time volume. Consequently, we first select the smallest value of C from Table I and solve Eq. (17) for d. We then verify that, for a physical error rate of $p = 10^{-3}$ and our calculated solution for d, the following holds for the corresponding p_{out} entry:

$$p_{\text{logical}} = 1 - (1 - p_C)^d > p_{\text{out}}. \tag{18}$$

If this condition is unmet, we must choose a better-performing T factory in the outer loop. We then proceed to the next entry on the list and repeat until we identify a T factory that performs at least as well as p_{logical}.

Once a specific T-factory is selected, the total number of allocated physical qubits across all modules at any given time and the total intramodular run-time (including the first subgraph preparation time at step 0, all consumption operations, and delays) can be calculated as follows:

$$\begin{aligned} &n_{\text{alloc-phys}} \\ &\quad = 4\tilde{n}_{\text{logical}} d^2 + n_{\text{per-leg}}^{\text{modules}}\left(n_{T\,\text{factories}} Q + 2d^2 l_{\text{transfer-bus}}\right), \\ &t_{\text{consump}}^{\text{total}} \\ &\quad = 8td\left(L_{\text{prep}}^0 + N_{\text{consump}}^{\text{seq}}\right) + t_{\text{distill}}^{\text{delay}} + t_{\text{prep}}^{\text{delay}}. \end{aligned} \tag{19}$$

In Eq. (19), t denotes the characteristic gate time, which sets the quantum tock of the architecture for all intramodular operations. The factor of 8 arises because, in our architecture, each surface-code sweep for stabilizer syndrome extraction circuits requires at most one initialization, two Hadamards, four entangling gates, and one measurement, all assumed to take time t (see, e.g., Ref. [7, fig. 9]). The temporal cost of initializing the first subgraph g_0 with $L_{\text{prep}}^0(S_{\text{prep}}^0)$ is always nonzero and has been explicitly included. FTQC performs the remaining graph preparations and T-state distillation operations in parallel with consumption operations. Overall delays may accumulate across time steps, defined as $t_{\text{distill}}^{\text{delay}} = \sum_i t_{\text{i}}^{\text{distill-delay}}$ and $t_{\text{prep}}^{\text{delay}} = \sum_i t_{\text{i}}^{\text{prep-delay}}$, which we detail below.

To execute each consumption subschedule i, a specific number of distilled magic states must be present on the T-transfer bus of each module. We must introduce a delay if we require more magic states than were accumulated on the T-transfer bus during the prior preparation step, $i-1$. The number of T states per module after a preparation step is

$$\begin{aligned} n_i^{T-\text{per-module}} &= \max\left(\left\lfloor \frac{n_{T\,\text{factories}} t_i^{\text{prep}}}{8tC} \right\rfloor, l_{\text{transfer-bus}}\right) \\ &\quad \text{for } i \in \{0, \ldots, n_{\text{widgets}} - 1\}. \end{aligned} \tag{20}$$

Here, we have defined the hybrid (intra- and intermodular) time required to prepare the subgraph for preparation step i as $t_i^{\text{prep}} = 8d(N_i^{\text{intra-comms}} t + N_i^{\text{cross-module}} t_{\text{inter}})$, where $N_i^{\text{intra-comms}} = \sum_{j \in L_{i,j}^{\text{prep}}} \left\lfloor \mathbb{L}_i^{\text{prep}} / \sum_{k \in L_{i,j}^{\text{prep}}} \lfloor d_{i,j,k}^{\max} / \tilde{n}_{\text{logical}} \rfloor \right\rfloor$

for $n_{\text{per-leg}}^{\text{modules}} > 1$ is the count of intramodule operations, while $N_i^{\text{cross-module}} = \sum_{j\in L_i^{\text{prep}}} \left\lceil \sum_{k\in L_{i,j}^{\text{prep}}} \lfloor d_{i,j,k}^{\max}/\tilde{n}_{\text{logical}} \rfloor / n_{\text{inter-pipes}} \right\rceil$ for $n_{\text{per-leg}}^{\text{modules}} > 1$ represents the number of cross-module operations in the preparation subschedule i (only in the vertical direction along a ladder leg). We assume that $t_{\text{inter}} \gg t$ is the characteristic physical timescale for all intermodule communications. Moreover, we define $d_{i,j,k}^{\max}$ as the maximum distance between any pairs of graph nodes in the tuple k of sublist j of preparation subschedule i (for details, see Ref. [44]), and $n_{\text{inter-pipes}}$ is the number of intermodular pipes connecting each pair of modules in the macroarchitecture.

We can now define the distillation delay for step $i \in \{0,\ldots,n_{\text{widgets}}-2\}$ as

$$t_{\text{i}}^{\text{distill-delay}} = \begin{cases} 8tC & \left\lceil \frac{L_{\text{consump},i}^{\text{max-seq}} - n_i^{T\,\text{per-module}}}{n_{T\,\text{factories}}} \right\rceil, \\ & \text{if } L_{\text{consump},i}^{\text{max-seq}} > n_i^{T\,\text{per-module}}, \\ 0, & \text{otherwise,} \end{cases}$$

where $L_{\text{consump},i}^{\text{max-seq}} = n_i^{\max-T} + L_\varepsilon n_i^{\max-R_Z}$ is the maximum sequential T length per module, considering all consumption nodes of step i. We have defined $n_i^{\max-T} = \max_j (n_{i,j}^T)$ for $i \in S_i^{\text{meas}}$, $j \in \{0,\ldots,n_{\text{per-leg}}^{\text{modules}}-1\}$ as the maximum number of T-basis measurement nodes in all modules specified by measurement subschedule S_i^{meas}. Similarly, $n_i^{\max-R_Z} = \max_j (n_{i,j}^{R_z})$ for $i \in S_i^{\text{meas}}$, $j \in \{0,\ldots,n_{\text{per-leg}}^{\text{modules}}-1\}$.

Similarly, for step i, an idle delay must be added if preparation for the next subgraph $i+1$ is not yet complete, and this delay runs in parallel across the interleaving modules in the other leg of the ladder. An upper bound on the individual time for intramodular node consumption in step i is $t_i^{\text{consump-intra}} = 8td(\lceil n_i^{\max-T}/n_{T\,\text{factories}} \rceil + L_\varepsilon \lceil n_i^{\max-R_Z}/n_{T\,\text{factories}} \rceil)$. Therefore, the preparation delay for step $i \in \{0,\ldots,n_{\text{widgets}}-2\}$ becomes

$$t_{\text{i}}^{\text{prep-delay}} = \begin{cases} t_{i+1}^{\text{prep}} & - t_i^{\text{consump-intra}} - t_{\text{i}}^{\text{distill-delay}}, \\ & \text{if } t_{i+1}^{\text{prep}} > t_i^{\text{consump-intra}} + t_{\text{i}}^{\text{distill-delay}}, \\ 0. & \text{otherwise.} \end{cases}$$

Lastly, we need to account for the total time required on the actual FTQC to stitch the output nodes of the graph at step i, located along one leg of the ladder, to the input nodes of the graph at step $i+1$, situated along the other leg of the ladder, for $i \neq n_{\text{widget}}-1$. We assume that we can perform the handover process *on-the-fly* by teleporting logical Bell pairs across the modules in a horizontal direction, which involves adding logical CNOT or CZ gates (or, for smaller problems, some equivalent SWAP operations). We must incur a cost if FTQC needs to teleport graph nodes vertically within the same leg (which is infeasible in $O(1)$ operations). Therefore, an upper bound for this quantity can be expressed as

$$t_{\text{handover-inter}}^{\text{total}} = 8t_{\text{inter}}d \sum_{i\in\{0,\ldots,n_{\text{widget}}-2\}} \left\lceil \frac{n_i^{\text{cross-module-I-O}}}{n_{\text{inter-pipes}}} \right\rceil, \tag{21}$$

where we define the total number of communications between cryostat modules needed to hand over output nodes of graph i to $i+1$ as $n_i^{\text{cross-module-I-O}} = \sum_{\tilde{o}\in S_{\text{output}}^i, \tilde{i}\in S_{\text{input}}^{i+1}} n_{\tilde{o},\tilde{i}}^i$; here, $n_{\tilde{o},\tilde{i}}^i$ is the number of crossings necessary to hand over node $\tilde{o}$ of graph i to node $\tilde{i}$ of graph $i+1$ for $i \neq n_{\text{widget}}-1$.

Finally, the total FTQC hardware time can be expressed as

$$t_{\text{hardware}}^{\text{total}} = t_{\text{intra}}^{\text{total}} + t_{\text{handover-inter}}^{\text{total}} + t_{\text{decode-delay}}^{\text{total}}. \tag{22}$$

In this equation, $t_{\text{decode-delay}}^{\text{total}}$ represents the overall delay incurred by performing decoding, entirely using classical methods, across all QEC cycles. We propose a method to estimate $t_{\text{tot}}^{\text{decode-delay}}$ in Appendix A, based on generic decoders.

The following summarizes the work flow as Algorithm 1.

ALGORITHM 1. The RRE estimation methodology based on the fixed logical architectures outlined in Sec. II.

input : Logical quantum algorithm, `QuantumCircuit`, U
input : Look-up table: T `factories` (C, Q, p_{out})
output: Total allocated physical qubits, $n_{\text{alloc-phys}} = 4\tilde{n}_{\text{logical}}d^2 + n_{\text{per-leg}}^{\text{modules}}(n_{\text{T-factories}}Q + 2d^2 l_{\text{transfer-bus}})$
output: Total intra-modular time, $t_{\text{intra}}^{\text{total}} = 8td(L_{\text{prep}}^0 + N_{\text{consump}}^{\text{seq}}) + t_{\text{distill}}^{\text{delay}} + t_{\text{prep}}^{\text{delay}}$

$\{\tilde{n}_{\text{logical}}, \tilde{S}_{\text{consump}}, \tilde{S}_{\text{prep}}, \tilde{S}_{\text{meas}}\}$ = WIDGETIZER-CABALISER-SCHEDULER (`QuantumCircuit`);
for *widget* $\in T$ *factories* **do**
  if *widget is (20-to-4)* **then**
    $n' \to 4n'$;
  while $M_\varepsilon ==$ *False* **do**
    Solve $\kappa d \left(\frac{p}{p_{\text{thresh}}}\right)^{(d+1)/2} (2\tilde{n}_{\text{logical}}\mathbb{L}_{\text{prep}}d + (2\tilde{n}_{\text{logical}} + n_{\text{per-leg}}^{\text{modules}} l_{\text{transfer-bus}})(N_{\text{consump}}^{\text{seq}}d + N_{\text{distill}}^{\text{seq}}C)) < -J$ for the smallest integer, d, where $J = \log(1/(1-p_{\text{algo-fail}}))$, $p = 10^{-3}$;
    $p_{\text{log-cell}} = 1 - (1 - \kappa(p/p_{\text{thresh}})^{(d+1)/2})^d$;
    $M_\varepsilon = p_{\text{logical}} > \epsilon$;
    if $M_\varepsilon ==$ *False* **then**
      Decrease ε;
  $M_{\text{logical}} = p_{\text{logical}} > p_{\text{out}}$;
  if $M_{log\text{-}cell} ==$ *True* **then**
    Break loop;

III. TEST ALGORITHMS

To demonstrate resource estimation using the graph-state formalism, we have generated circuits for selected pedagogical quantum algorithms that still target core computational challenges in real-world applications. Resource-efficient compilation necessarily involves hardware considerations in the synthesis methodology and the empirical evaluation of alternatives when analytical results are not readily available. As such, these circuits are then compiled into FT quantum computing patterns tailored to superconducting architectures presented in Sec. II. This section summarizes the test cases selected for circuit synthesis to demonstrate our resource-estimation methodology.

A. Quantum Fourier transform

The quantum Fourier transform (QFT) is a common component of quantum algorithms, including quantum phase estimation. From the perspective of hardware-specific resource analysis, QFTs are likely to feature in application-relevant resource analysis, yet can be scaled down to a small handful of logical input gates, allowing us to trace and debug compilation work flows in our tool chain at the level of logical operations. This makes QFT circuits ideally suited as the first test cases of our estimation framework. This work analyzes QFT circuits on n_{input} logical qubits, where $n_{\text{input}} = 4, 10, 25, 50, 75, 100, 250, 500,$ or 1000.

B. Hamiltonian simulations

Simulation of lattice-based Hamiltonians encompasses a family of critical applications investigated by researchers in quantum chemistry and condensed-matter physics. Moreover, simulating dynamics may be more useful for smaller hardware systems than methods that evaluate static properties, such as the ground state.

Hamiltonian simulation consists of evolving an initial state $|\psi_0\rangle$ to a state at time t, $|\psi\rangle$, according to

$$|\psi\rangle = e^{-iHt}|\psi_0\rangle, \tag{23}$$

where H is the Hamiltonian of the system of interest. We consider the simulation of two families of Hamiltonians: 2D transverse-Ising models and 2D Fermi-Hubbard models. Each of these will be described in more detail below.

1. Transverse-Ising Hamiltonian simulations

The Los Alamos National Laboratory (LANL) has created a public repository containing circuit generation and problem analysis for problems that may be amenable to quantum computing and are of interest to the scientific community [72]. Drawing inspiration from their investigation of magnetic lattices, we perform resource estimation for the simulation of a time-invariant transverse-Ising Hamiltonian,

$$H = \sum_{\langle \mathbf{jk}\rangle} Z_{\mathbf{j}} \otimes Z_{\mathbf{k}} + 0.1 \sum_{\mathbf{j}} X_{\mathbf{j}}. \tag{24}$$

Here, $X_{\mathbf{i}}$ and $Z_{\mathbf{i}}$ represent the Pauli X and Z operators, respectively, acting on lattice site $\mathbf{i}$, and the angle brackets denote lattice sites connected by an edge.

Based on the studies documented by LANL, we have carried out resource estimates of simulation of the Hamiltonian in Eq. (24) for a total time of $t = 1$ (in the inverse Hamiltonian units) on a 2D triangular lattice (sometimes called a *2D hexagonal Bravais lattice*) of size $N \times N$, where N is an integer between 2 and 19.

2. Fermi-Hubbard Hamiltonian simulations

We have also investigated the Fermi-Hubbard simulation as a core computational capability for solving problems in room-temperature superconductivity [73]. These problems typically involve studying the ground-state energy, state configurations, and other observable system properties, including time-dependent quantities.

The Fermi-Hubbard Hamiltonian we consider is given by

$$\begin{aligned} H = &-J \sum_{\langle \mathbf{j},\mathbf{k}\rangle,\sigma} \left[c^{\dagger}_{\mathbf{j},\sigma} c_{\mathbf{k},\sigma} + c^{\dagger}_{\mathbf{k},\sigma} c_{\mathbf{j},\sigma} \right] \\ &+ J' \sum_{\langle\langle \mathbf{j},\mathbf{k}\rangle\rangle,\sigma} \left[c^{\dagger}_{\mathbf{j},\sigma} c_{\mathbf{k},\sigma} + c^{\dagger}_{\mathbf{k},\sigma} c_{\mathbf{j},\sigma} \right] \\ &+ U \sum_{\mathbf{j}} n_{\mathbf{j},\uparrow} n_{\mathbf{j},\downarrow} + \frac{h_z}{2} \sum_{\mathbf{j}} (n_{\mathbf{j},\uparrow} - n_{\mathbf{j},\downarrow}) \\ &+ \mu \sum_{\mathbf{j},\sigma} n_{\mathbf{j},\sigma}, \end{aligned} \tag{25}$$

where the single and double angled brackets denote all nearest-neighbor and next-nearest-neighbor lattice site pairs, respectively. $c^{\dagger}_{\mathbf{j},\sigma}(c_{\mathbf{j},\sigma})$ creation (annihilation) operators of a spin-σ fermion on the $\mathbf{j}$th lattice site where $\sigma \in \{\uparrow, \downarrow\}$. J (primed or unprimed) and U capture the magnitudes of the hopping and repulsion energies, respectively. $n_{\mathbf{k},\sigma} = c^{\dagger}_{\mathbf{k},\sigma} c_{\mathbf{k},\sigma}$ is the number operator, h_z is an external applied magnetic field, and μ is the chemical potential.

To motivate the specific Fermi-Hubbard simulations for which we will obtain resource estimates, we have reviewed the literature to identify configurations of interest to the community. Broadly speaking, methods for simulating a Fermi-Hubbard system can be classified as stochastic or deterministic. As mentioned in Ref. [74], deterministic methods do not "suffer from numerical issues seen in stochastic simulations when dealing with systems with nonzero chemical potential." Moreover, this instability due

to the "sign problem" becomes prohibitive for moderate $\mu \sim 0.3$. This suggests separating instances into those with $\mu = 0$ and those with $\mu > 0.3$. Here, we reference a few exemplary studies in the literature on stochastic and deterministic methods.

Some examples of stochastic studies include a study using auxiliary-field quantum Monte Carlo (AFQMC) to estimate ground states on lattice sizes of 4×4 to 16×16, $U/J = 4, 8$, and $J' = 0$. [75]. In addition, Ref. [76] has used determinant quantum Monte Carlo (DQMC) to study doped ground-state estimation using $U/J = 6$, $J'/J = -0.25$, and lattice sizes of 8×8, 12×8, 12×12, 16×8. In contrast, examples of deterministic studies include work on numerical linked-cluster expansions [77]. In this study, researchers selected random repulsion strengths from an uncertainty "box" about U_0. Simulations were for a periodic 10×10 lattice with $U_0/J = 8$, $J' = 0$. Other researchers have used projected density-matrix embedding (PDME) to perform a 2D Fermi-Hubbard study with $U/J = 2, 4, 6, 8$, $J' = 0$ on a 40×40 lattice [78]. Finally, in Ref. [74], the authors use projected entangled-pair states (PEPSs) with a bond dimension of $D = 10$, and claim that accuracy increases with D. Most of their results are from a 3×4 honeycomb lattice with $U/J \in [0, 6]$ and $J' = 0$. Still, the authors have been able to calculate the ground state via imaginary time evolution on a 15×15 bipartite honeycomb lattice.

Of note is that in Ref. [74] it has been determined that the time required to run these studies on a CPU varies as $t_{\text{CPU}} \propto L_x L_y (2\chi^3 D^4 d^2 + \chi^2 D^2 d^2 + \chi^2 D^4 d^4)$, where χ is the truncated bond dimension and d is the problem dimension. Memory is scaled as $2\chi^2 D^4 d^2 M_{\text{num}} + 2N\chi^2 D^2 d M_{\text{num}}$. The authors have empirically found that $\chi = 3D$ is sufficiently accurate for their studies, with an error of about 1%. These estimates provide a useful guide post for the tipping point at which FTQCs may outperform classical computers on these types of problems.

Based on the parameters and layouts used in the Fermi-Hubbard Hamiltonian studies [74–78], we adopted layouts and parameters for our simulations consisting of a 2D square lattice for total time $t = 1$, with $J = 1$, $J' = 0$, $h_z = 0$, $\mu = 0$, $U = 2$, and lattice sides of length $N = 2$ to 20, 30, 50, 75, and 100.

3. Quantum algorithm and approaches for Hamiltonian simulations

Owing to its optimal scaling characteristics for queries to the block encoding of the Hamiltonian [15], we have chosen quantum signal processing (QSP) as the Hamiltonian-simulation algorithm for which we obtain a resource estimate. QSP is briefly described below, but more details can be found in Refs. [15,79–81].

Given a Hamiltonian H and simulation time t, QSP performs a sequence of single-qubit rotations, parametrized by a sequence of *phase angles*, interlaced with a quantum-walk-like operator. This allows one to apply a polynomial approximation of e^{-iHt} to the input target state. The sequence of phase angles determines the polynomial approximation. It is computed offline, using a classical algorithm, with the desired error tolerance specified by the user-defined parameter ϵ_{qsp}. ϵ_{qsp} is also inversely related to the probability of success of the QSP algorithm. Thus, for an accurate simulation, with a high probability of success, ϵ_{qsp} may have to be set relatively small. For implementation details, we follow the examples accompanying the pyLIQTR [82] package (v1.3.3), as well as the work done in Refs. [83,84].

For demonstration purposes, we set the desired probability of success of our algorithm, p_{algo}, to be equal to 0.95. In real-world applications, this is typically fixed to meet outcome requirements (e.g., accuracy and wall time). Denoting the probability of success of a QSP circuit, p_{qsp}, we therefore require that

$$p_{\text{algo-fail}} \leq p_{\text{qsp}}. \tag{26}$$

Since circuit size is inversely related to ϵ_{qsp}, it is prudent to choose ϵ_{qsp} to be as large as possible while still guaranteeing that p_{qsp} exceeds the algorithm target of 0.95. According to Ref. [15], $(1 - 2\epsilon_{\text{qsp}}) \leq p_{\text{qsp}}$. Thus, to ensure that Eq. (26) is satisfied, we set

$$1 - 2\epsilon_{\text{qsp}} = p_{\text{algo-fail}}, \tag{27}$$

and so in this work we choose ϵ_{qsp} to be equal to 0.025.

C. Simulation-circuit synthesis

For our Hamiltonian-simulation circuits, we have used pyLIQTR v1.3.3 to construct the Hamiltonians and synthesize the QSP circuits. The pyLIQTR package provides tools that construct the Hamiltonians, calculate the necessary phase angles for QSP simulation, and generate QSP circuits. In addition, pyLIQTR allows the user to request random phase angles rather than solving for the precise values required for an accurate simulation. In our work, we have used random phase angles. This is appropriate because we only perform resource estimation, not numerically accurate simulations. The correct *number* of phase angles is used in the circuits but their numerical *values* are random. This provides a conservative overestimate of the resource estimations without explicitly incurring the time-consuming step of solving for the phase angles. For Fermi-Hubbard models, we have utilized the pyLIQTR `FermiHubbard` model class, and employed the `FermiHubbardSquare` block encoding, which is based on Ref. [85].

The same overall approach has been taken for the time evolution of the transverse-Ising models. However, for these problems, we have used the pyLIQTR

TABLE II. The RRE results on a fixed superconducting fault-tolerant quantum computing architecture (see Sec. II), with space-optimized graph-state compilation. Here, "Number of logical bus patches" includes patches for the memory and auxiliary buses that mediate logical measurements in the bilinear pattern shown in Fig. 1. Likewise, "Number of T patches" includes logical patches for all T factories and T-transfer buses.

Input circuit		Logical resources		Superconducting FTQC resources					
Instance	$\tilde{n}_{\text{logical}}$	Number of logical bus patches	Number of T patches	Total allocated physical qubits, $n_{\text{alloc-phys}}$	Total modules, $2n^{\text{modules}}_{\text{per-leg}}$	$t^{\text{total}}_{\text{hardware}}$	$P_{\text{Diss,4 K}}$	$P_{\text{Diss,20 mK}}$	E_{tot}
Fermi-Hubbard									
2×2	36	280	4.05×10^3	1.32×10^6	2	4.09 s	840 W	168 nW	668 Wh
4×4	74	332	3.32×10^3	1.21×10^6	2	35.1 s	840 W	168 nW	5.73 kWh
8×8	182	728	2.07×10^3	1.38×10^6	2	9.49 m	840 W	168 nW	93 kWh
16×16	578	2.31×10^3	1.25×10^3	3.30×10^6	4	6.9 h	1.68 kW	336 nW	8.11 MWh
20×20	878	3.52×10^3	1.4×10^3	4.96×10^6	6	37.8 h	2.52 kW	504 nW	66.7 MWh
50×50	5.05×10^3	20.2×10^3	5.54×10^3	36.1×10^6	42	7.14 y	17.6 kW	3.53 μW	772 GWh
100×100	10.1×10^3	40.7×10^3	19.3×10^3	114×10^6	132	63.3 ky	55.4 kW	11.1 μW	21.5 TWh
QFT									
N-4	4	88	6.38×10^3	1.10×10^6	2	2.8 ms	840 W	168 nW	457 mWH
N-10	10	248	5.5×10^3	1.29×10^6	2	41.7 ms	840 W	168 nW	6.81 Wh
N-25	25	2.02×10^3	3.75×10^3	1.31×10^6	2	427 ms	840 W	168 nW	69.7 Wh
N-100	100	3.39×10^3	2.76×10^3	3.18×10^6	4	10.4 s	1.68 kW	336 nW	3.39 kWh
N-500	500	18.4×10^3	4.5×10^3	17.5×10^6	20	2.03 m	8.4 kW	1.68 μW	199 kWh
N-1000	1×10^3	37.2×10^3	9.57×10^3	43.4×10^6	52	10.6 m	21.8 kW	4.37 μW	2.69 MWh
Transverse Ising									
2×2	67	652	2.43×10^3	1.41×10^6	2	64.4 s	840 W	168 nW	10.5 kWh
5×5	212	2.98×10^3	816	4.39×10^6	6	6.64h	2.52 kW	5.04 nW	6.42 MWh
10×10	654	10.1×10^3	2.78×10^3	16.7×10^6	20	5.06 d	8.4 kW	1.68 μW	714 MWh
19×19	2.21×10^3	41.1×10^3	13.5×10^3	87.1×10^6	110	82.6d	46.2 kW	9.24 μW	64.1 GWh

`Heisenberg` model class, and `PauliLCU` block encodings. For QFT instances, we have used the QFT circuits provided by the `qft` method in CIRQ.

IV. RESOURCE-ESTIMATION RESULTS

The key physical-resource requirements for the test cases described in Sec. III across various problem sizes are summarized in Table II. We demonstrate that our tool, RRE, can estimate resource requirements for problem sizes consistent with the literature and that extend beyond the scales of current studies.

Our methodology enables us to extend the state of the art in several ways: (1) a full accounting for resources for both T counts, but also for executing the Clifford portions of the circuit as well; (2) a breakdown of physical and temporal resources allocated to portions of the execution of the algorithm, namely, Clifford, T state, and teleportation between modules; and (3) a quantitative estimation of the impact on total resources of the performance of subsystems.

When we visualize this ensemble of compiled logical resources and physical-resource requirements (Figs. 7 and 8, respectively), we observe trajectories of run-time and auxiliary bus size that increase with problem size. This provides validation of the reasonableness of our methodology and its utility as a tool to support future studies. Interestingly, we observe a decreasing trend in the number of logical qubits for small-sized problems until the number of modules increases. As the number of modules increases, we observe a sharp rise in the number of logical qubits. This is because physical qubits are assigned to T factories rather than to logical memory qubits or the auxiliary bus. As the problem size grows, the number of physical qubits used in the bilinear quantum memory and auxiliary bus pattern increases in the single-module regime. This reduces the number of T factories that can be accommodated within the module. As the T factories are large monolithic regions of logical qubits, removing them gives a net reduction in the number of logical qubits in use, even though the number of patches in the bilinear pattern is larger. The available storage space in a room can decrease even as smaller objects are added, provided that those smaller objects require removing larger objects to accommodate them.

The full accounting of both Clifford and T-state contributions to total resources, (1), is shown in Fig. 9 for all families of applications for which we have estimated resources. It is interesting to note that conventional T-counting methodologies do *not* often reveal resource

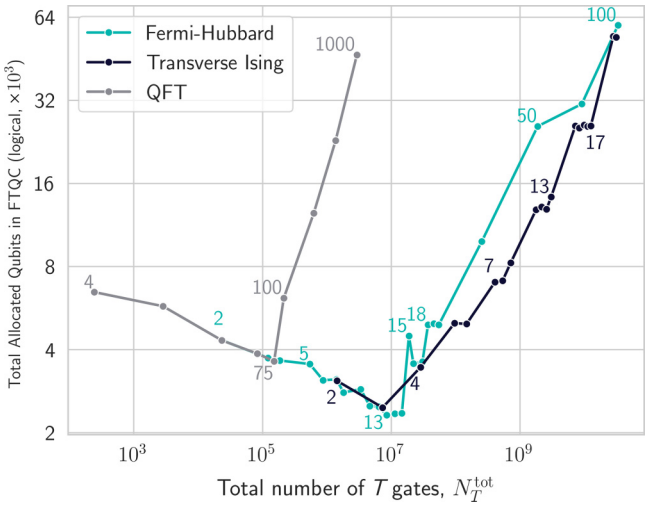

FIG. 7. The total allocated logical qubits (in the bilinear pattern) and T count for three sets of test cases. Gray indicates quantum Fourier transform (QFT) results for sizes 4 to 1000. The black curve shows the Hamiltonian evolution of the transverse-Ising model on triangular lattices ranging from 2×2 to 19×19. The teal line shows estimates from Hamiltonian simulations of a Fermi-Hubbard model on a square lattice, with sizes ranging from 2×2 to 100×100.

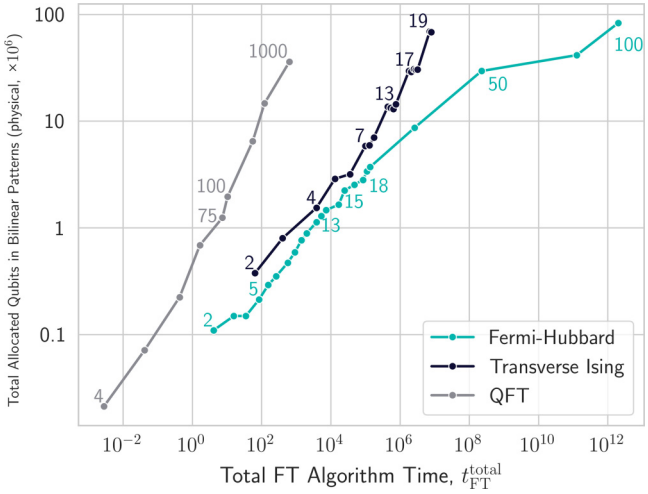

FIG. 8. The total allocated physical qubits (in the bilinear pattern) and FT hardware run-time for three sets of test cases. Gray indicates QFT results for sizes 4 to 1000. The black curve shows the Hamiltonian evolution of the transverse-Ising model on triangular lattices ranging from 2×2 to 19×19. The teal line is an estimate for the Hamiltonian simulation of a Fermi-Hubbard model on a square lattice with sizes ranging from 2×2 to 100×100. We set the Hamiltonian evolution time to 1 in units of $1/J$ for both the transverse-Ising and Fermi-Hubbard models.

competition between the Clifford and T-state generations of a quantum architecture. As problems scale, physical qubits must be allocated to meet the increased requirements of Clifford processing, thereby reducing the number of available physical qubits for MSD. This, in turn, reduces the number of T factories that can be applied to a calculation, thereby increasing run-time in a nonlinear fashion.

This impact on the breakdown of the total run-time, (2), is evident in Fig. 10. Here, the contribution to graph consumption run-time (using distilled T states) increases dramatically for the largest applications considered. This is because fewer physical qubits are allocated to create higher-precision T states, increasing the run-time penalty.

The impact of run-time on teleportation in the distributed architecture is also evident in this figure. Despite a relatively slow intermodule interaction time (5 times slower than the native code cycle), the contribution to overall run-time is application dependent. For QFT and transverse-Ising applications, a relatively small portion of the overall run-time is spent performing this intermodule interaction, even at large problem sizes. This shows that computation can be partitioned across specific architectures and algorithms to enable distributed computing with relatively low impact. However, this appears to be algorithm dependent, and for Fermi-Hubbard problems, the penalty for distributed quantum computation is high.

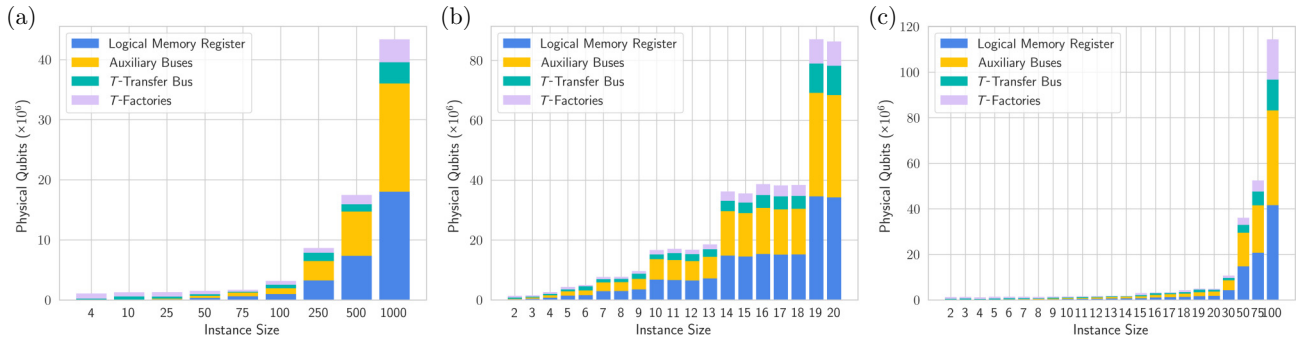

FIG. 9. Physical-qubit allocations for the types of logical qubits described in the logical microarchitecture shown in Fig. 1; those involved in the logical memory register, auxiliary buses (for single-qubit measurements, queuing, and routing), the T-transfer bus, and T factories—(a) QFT; (b) transverse-Ising; (c) Fermi-Hubbard.

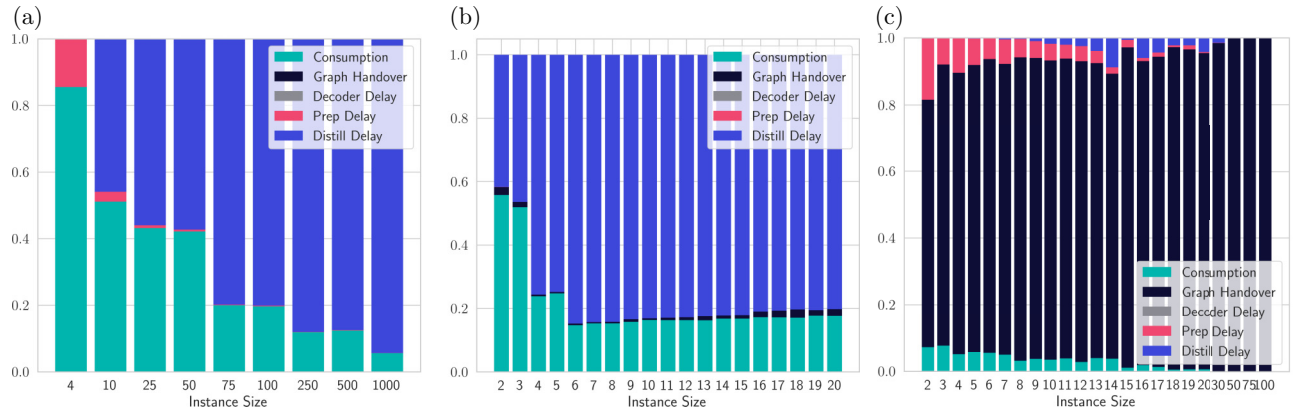

FIG. 10. The proportion of the FTQC compute time allocated to graph-state consumption (using distilled T states), intermodule communications, distillation, and decoding delays for time-evolution problems—(a) QFT; (b) transverse-Ising; (c) Fermi-Hubbard.

Moreover, we can test the sensitivity of run-time to parameters in this architectural model (3). In Fig. 11, we select an instance from each family, vary the number of intermodule connections, and calculate the impact on run-time. We define this in terms of a *pipe* where the number of connections equals the required code distance for the algorithm. Additional pipes will enable parallel node-to-node teleportation between widgets. This is done to quantify the penalty associated with not having a fully connected lattice of physical qubits between modules. Here, we observe that while run-time increases with fewer intermodule connections, the marginal benefit of adding more connections diminishes. Once again, this impact is highly application dependent.

Similarly, we can directly quantify the impact of changing the decoders on the physical system size. Results for Fermi-Hubbard instances of increasing problem size are shown in Fig. 12 and demonstrate the utility and flexibility of our resource-estimation tool for assessing the effects of changes across multiple subsystems within the overall architecture.

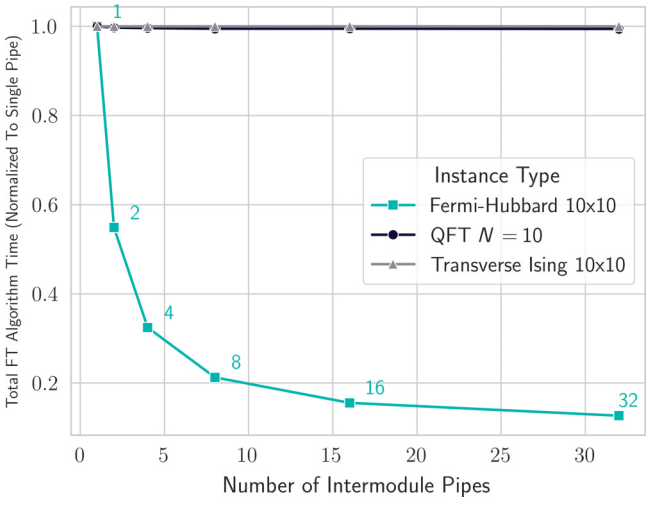

FIG. 11. A comparison of the normalized total FT run-time and the number of intermodule pipes. The number of intermodule pipes varies from 1 to 32. The run-time is normalized to the corresponding value for one pipe. Black indicates QFT results of size 10. The gray plot is the Hamiltonian evolution of the transverse-Ising model on a triangular lattice of size 10. The teal line represents estimates for the Hamiltonian simulation of a Fermi-Hubbard model on a 10×10 square lattice.

V. FUTURE WORK

The results from Sec. IV can be improved by focusing on at least the following two aspects: (a) using a decoder that is faster and may have better scaling than the standard

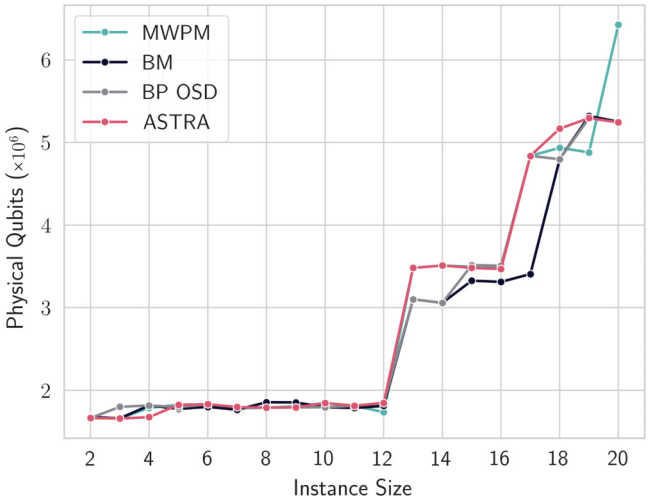

FIG. 12. The total number of allocated physical qubits (parameter 35 of Appendix A) for Hamiltonian evolution of the Fermi-Hubbard of various sizes. The line colors denote different decoders; teal indicates MWPM, black is the belief-matching (BM) decoder, gray is belief propagation with ordered statistics decoding (BP OSD), and magenta is ASTRA [86].

choice of MPWM; and (b) optimizing the widget circuits by, e.g., reducing circuit gate counts and depths. Below, we present the savings potential of these two approaches, along with additional details on the design of high-cooling-power cryostats.

A. Replacing the MWPM decoder

The MWPM decoder (e.g., Ref. [87]) is the *de facto* standard for resource estimation. Other decoders have been proposed, but MWPM remains the most efficient with respect to the decoding speed−physical error rate trade-off: it can handle high error rates while maintaining polynomial complexity. There are faster lower-polynomial-complexity decoders (e.g., union-find variants or pure belief propagation) and decoders with linear complexity, which are challenging to scale to high distances (e.g., machine-learning decoders). Nevertheless, no alternative decoder exists that has, at the same time, a lower degree of polynomial decoding complexity and a threshold higher than MWPM.

Herein, we report results in Fig. 13 for a machine-learning decoder that outperforms MWPM: it has a higher threshold and linear run-time complexity. Our machine-learning decoder, ASTRA [86], uses graph neural networks (GNNs) in a novel way: we learn a message-passing algorithm on the Tanner graph of the code. We can train such decoders within hours on gaming GPUs for surface codes at distances up to 11. In contrast to other machine-learning decoders, the training time of our decoder is significantly lower (see, e.g., Ref. [88]) for comparable depths.

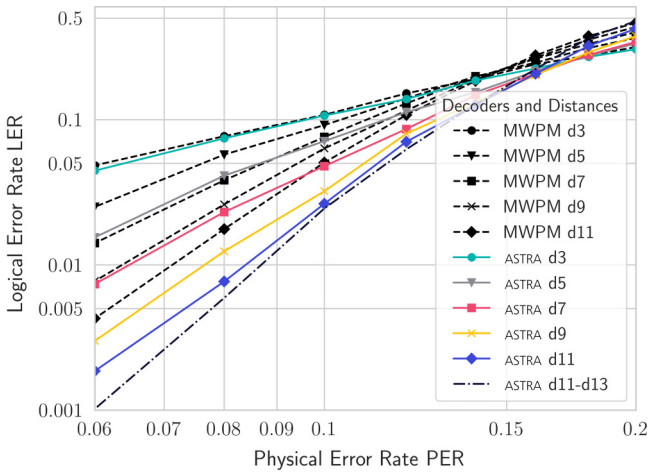

FIG. 13. The logical error rate (LER) for the code-capacity depolarizing noise of ASTRA versus MWPM. ASTRA has a threshold of approximately 17% and MWPM has a threshold of approximately 14%. ASTRA clearly outperforms MWPM in terms of LER. In particular, ASTRA d9's results are better than those of MWPM d11.

We train distinct GNN decoders for each distance using syndromes generated by passing the surface codes through a code-capacity noise channel. This channel assumes that the physical data qubits are affected by noise with a probability p and that the syndrome measurements are perfect. For this work, we do not report the decoding of circuit-level correlated noise (see, e.g., Ref. [87]) and we focus entirely on correcting depolarizing noise.

After training our decoders, we evaluate their decoding performance numerically and capture the performance of the decoder through an equation of the type from Eq. (1):

$$p[\mathrm{GNN}] = 0.56\left(\frac{p}{0.17}\right)^{\frac{d+1}{2}}. \tag{28}$$

Typically, the performance-scaling equations for the surface code are obtained from simulations with circuit-level noise [e.g., Eq. (1)] because this is considered the most accurate model. Nevertheless, the code capacity is more straightforward to benchmark for preliminary decoder versions. To enable comparison with ASTRA, we also benchmark MWPM with code-capacity noise. The results are illustrated in Fig. 13. Based on the MWPM code-capacity errors, we obtain the MWPM scaling equation as

$$p[\mathrm{MWPM}] = 0.52\left(\frac{p}{0.14}\right)^{\frac{d+1}{2}}. \tag{29}$$

We observe that ASTRA has a higher threshold than MWPM (17% > 14%) and provides faster (better) logical error suppression (0.56 > 0.52). This implies that by replacing the MWPM with ASTRA, one can achieve the same logical error rate with lower-distance surface codes. After benchmarking its performance under circuit-level noise, future work will focus on quantifying the exact savings achieved by using the new decoder.

B. Ultra-large-scale circuit optimization

The Fermi-Hubbard circuit instances analyzed in the previous sections are, for practical regimes, too large in terms of gate counts. Compiling, analyzing, and optimizing such circuits at graph-state or Clifford + T level is challenging: the state-of-the-art quantum software tools cannot handle ultra-large-scale circuits.

Widgets (detailed in Appendix B) are subcircuits representing relevant parts of the larger circuits. Due to their size, widgets can be more easily analyzed and optimized. Nevertheless, even widgets are complicated to optimize for metrics such as T and T depth. Optimization is a highly complex task, and most optimization heuristics are not designed (or do not perform as expected) for (sub)circuits of tens of logical qubits and hundreds of gates.

Tools such as `pyLQTR` are built on top of other quantum software frameworks, such as Google CIRQ, which were

TABLE III. Widget optimization results: addition circuits from Maslov [91] are optimized for T count. The original `T-count` is reported in the second column and the optimized `T-count` in the third column. The optimization of Adder2048 required approximately 1 h.

Circuit	Original `T-count`	Optimized `T-count`	Improvement (%)
Adder8	266	236	11.27
Adder16	602	536	10.96
Adder32	1274	1144	10.20
Adder64	2618	2368	9.54
Adder128	5306	4648	11.64
Adder256	10 682	9258	13.33
Adder512	21 434	18 562	13.39
Adder1024	42 938	38 216	10.99
Adder2048	85 946	76 056	11.50

not designed for extreme scalability. We bridge this gap by developing and implementing a software tool that can easily handle ultra-large-scale circuits. Our tool, PANDORA [89], is based on PostgreSQL and leverages the capabilities of modern relational-database software supporting multithreaded operations on quantum circuits.

PANDORA provides a simple API for developing new and more complex optimization methods. We have interfaced our tool with Google QUALTRAN [90] and can easily extract large circuits for analysis and optimization. Currently, the circuits and widgets available in QUALTRAN have been manually optimized; therefore, automatic optimization is even more challenging. Therefore, the results from Table III are far from optimal circuits.

Table III reports preliminary results on optimizing arithmetic widgets as presented in Ref. [91]. The latter are ubiquitous building blocks of practical application circuits, such as the Fermi-Hubbard model. Our circuit-optimization procedure is not as complex as the one described in Ref. [91]. We have implemented a heuristic that uses multiple threads for applying circuit template rewrites [91]: (1) decomposition of gates (e.g., *TOFFOLI*); (2) canceling H, T, and X gates; (3) canceling CNOT gates; (4) commuting gates to the left and the right; and (5) reversing directions of CNOT gates.

To conclude, quantum software for ultra-large-scale circuit analysis and optimization can potentially lower the resources required for practical quantum computing applications. PANDORA is a first step toward optimizing at scale.

C. Cryostat design

The detailed design of high-cooling-power cryostats for operation at this scale is a significant challenge that will require dedicated effort over the coming years. For the resource estimates in this paper, we have produced total dissipation power (TDP) estimates for both the 4-K and 20-mK stages. These are presented in columns $P_{\mathrm{Diss,4\,K}}$ and $P_{\mathrm{Diss,20\,mK}}$ in Table II, respectively. While the numbers appear daunting by today's standards, several facilities provide well in excess of kilowatts of cooling power at 2 K, such as the Linear Coherent Light Source II (LCLS II) at the Stanford Linear Accelerator Center (SLAC). This system can produce 4 kW of cooling power at 2 K and 18.5 kW at 4.5 K. While we consider a design for such a system to be beyond the scope of the current work, we believe it is feasible within the timescales required to develop it.

VI. CONCLUSIONS

This paper proposes a fault-tolerant architecture based on modular quantum processing units that uses superconducting qubits, the surface code, and coherent intermodule connections. We provide quantitative estimates for the scale and run-time of various quantum computations. Our software tool compiles large-scale quantum circuits into a graph state, schedules them on a model of our logical architecture, and, from this, estimates the physical resources and time required for test cases derived from the existing literature. This model-driven approach delineates *how* a superconducting platform could accomplish the target computation, including details on the scheduled execution of graph-state preparation and consumption, as well as hardware-level justifications for some of the most critical size and power considerations. We have demonstrated that such a fault-tolerant quantum computer could perform scientifically interesting computations, such as a 20×20 Fermi-Hubbard time evolution, using 5.2×10^6 physical qubits in less than two days. However, the value of such a computation remains an open question. When utilized in the context of a complete high-temperature superconductivity study, the number of computations and their probability of success increase the estimated computation time to the order of years [73]. More work is needed to co-optimize this FTQC architecture with algorithms of interest, with distributed computation across multiple modules presenting as the major bottleneck in this study.

This work has enabled research toward the development of real quantum computing platforms that provide the advantages of fault-tolerant operations. It serves as a software test bed for evaluating the impact of intrinsic algorithmic, compilation, decoding, and physical-layer proposals, as well as system-architecture proposals. Moreover, this allows for *direct* per-algorithm comparison of different modalities by assessing run times, spatial, and energy resource costs using the presented RRE software. Rapid iteration in a simulated environment is a proven method for accelerating technological development, and we hope that our efforts to develop this tooling contribute positively to this endeavor.

ACKNOWLEDGMENTS

The views, opinions, and/or findings expressed are those of the authors. They should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. This research was developed with funding from the Defense Advanced Research Projects Agency under Agreement No. HR00112230006. J.R. was supported by the Sydney Quantum Academy, Sydney, New South Wales, Australia.

DATA AVAILABILITY

The data that support the findings of this paper are not publicly available because they contain commercially sensitive information. The data are available from the authors upon reasonable request.

APPENDIX A: GLOSSARY OF DEFAULT RRE OUTPUTS

Building on the work flow of the graph-state-based fault-tolerant resource-estimation method outlined in the main text, we summarize and reiterate the selected output parameters of RRE, providing additional details where needed. We detail 49 logical architecture parameters or hardware-relevant outputs, which RRE writes to `stdout` and `CSV` files by default, and explain how to calculate them. The delivered `CSV` file can include results for additional output parameters. Users can add custom quantity classes for estimation in RRE using class templates from the `resources.py` module.

(a) **Parameter 1: Code distance, d**

d denotes the surface-code distance of logical patches. For the surface code, $\lfloor (d-1)/2 \rfloor$ errors can be corrected before a logical operation fails. All nondistillation logical patches have size $2d^2$ and serve as foundational elements for architectural components in the FTQC (T factories are parametrized by their own code-distance sets).

We need to calculate the minimum viable d precisely and iteratively according to Algorithm 1, which requires satisfying a logarithmic inequality in step 6, also referred to in Eq. (17).

(b) **Parameter 2: Number of logical qubits in the quantum memory register, $\tilde{n}_{\text{logical}}$**

The term $\tilde{n}_{\text{logical}}$ refers to the maximum number of logical nodes needed to process the complete graph state $\mathcal{G}$ according to the widgetization-based set $\tilde{S}_{\text{est}}$ at any given time. In other words, this represents the maximum quantum memory required. Each complete quantum memory register chain, shared among the modules on the same leg of the macroarchitecture, contains $\tilde{n}_{\text{logical}}$ logical qubits. To prevent any quantum memory bottlenecks or delays, we allocate a fixed quantum memory and auxiliary bus space of size $2\lceil \tilde{n}_{\text{logical}}/n_{\text{modules}}^{\text{per-leg}} \rceil$ to all modules and steps of the schedules. There may be more efficient methods for assigning the bilinear quantum register and auxiliary bus spaces.

Note that the maximum quantum memory required for the complete U is a distinct quantity, n_{logical}. For widgetized circuit input, we derive a proxy for this quantity from the subgraphs as $\tilde{n}_{\text{logical}} = \max(n_{\text{logical}}^{g_0}, \ldots, n_{\text{logical}}^{g_{n_{\text{widgets}}-1}}) \approx n_{\text{logical}}$. CABALISER can efficiently output each n_{logical}^{g} for sufficiently small graph states.

(c) **Parameter 3: Number of T factories per module, $n_{T\,\text{factories}}$**

The number of Litinski-style T factories per module is calculated as $n_{T\,\text{factories}} = \lfloor (l_{\text{edge}}-1)/\lfloor l_{\text{width}}/2d^2 \rfloor \rfloor n_{T\,\text{factories}}^{\text{col}}$. The factories send purified magic states to the T-transfer and auxiliary buses required for measurements on the logical qubits during the consumption stage. For more details, see Sec. II D 7.

(d) **Parameter 4: Number of *logical memory* qubits per module**

The number of logical memory, or graph-state, or data qubits in the bilinear pattern in every module is equal to $\left\lceil \tilde{n}_{\text{logical}}/n_{\text{modules}}^{\text{per-leg}} \right\rceil$. These correspond to the blue tiles in Fig. 1. In other words, every module on a leg of the macroarchitecture contains the same number of logical memory qubits. Although the last module may require fewer active nodes for quantum operations, the same space is always allocated to this component.

(e) **Parameter 5: Number of *physical memory* qubits per module**

The number of memory, or graph-state, or data physical qubits in the bilinear pattern in every module is equal to $2d^2\lceil \tilde{n}_{\text{logical}}/n_{\text{modules}}^{\text{per-leg}} \rceil$.

(f) **Parameter 6: Number of *logical* auxiliary qubits in the auxiliary bus per module**

The number of logical auxiliary qubits in the auxiliary bus section of each module is equal to $\left\lceil \tilde{n}_{\text{logical}}/n_{\text{modules}}^{\text{per-leg}} \right\rceil$. These correspond to the yellow tiles in Fig. 1, which FTQC uses to facilitate logical measurements on the logical graph-state qubits. Each module on a leg of the macroarchitecture contains the same number of logical auxiliary qubits. Although the last module may need fewer active nodes for quantum operations, the same space is always allocated for this component.

(g) **Parameter 7: Number of *physical* qubits in the auxiliary bus per module**

The number of physical qubits in the auxiliary bus portion of every module is equal to $2d^2\left\lceil \tilde{n}_{\text{logical}}/n_{\text{modules}}^{\text{per-leg}} \right\rceil$.

(h) **Parameter 8: Number of *logical* qubits in the *T*-transfer bus per module**, $l_{\text{transfer-bus}}$

The number of logical qubits in the T-transfer bus portion of every module is calculated as $l_{\text{transfer-bus}} = (l_{\text{edge}} - l_{\text{qbus}} - n^{\text{col}}_{T\,\text{factories}} \left\lceil L_{\text{length}}/\sqrt{2}d \right\rceil) l_{\text{edge}} + n^{\text{col}}_{T\,\text{factories}} \left\lceil L_{\text{Length}}/\sqrt{2}d \right\rceil$. These correspond to the teal tiles in Fig. 1, which FTQC uses to actively store and transfer purified T states from MSD factories to the auxiliary bus.

(i) **Parameter 9: Number of *physical* qubits in the *T*-transfer bus per module**

The number of physical qubits in the T-transfer bus portion of each module is $2d^2 l_{\text{transfer-bus}}$.

(j) **Parameter 10: Number of logical qubits allocated to all *T* factories in each module**

The number of logical nodes assigned to all $n_{T\,\text{factories}}$ T factories in each module can be calculated as $n_{T\,\text{factories}} \left\lceil L_{\text{length}}/(\sqrt{2}d) \right\rceil \left\lceil L_{\text{width}}/(\sqrt{2}d) \right\rceil$. These correspond to the purple tiles in Fig. 1.

(k) **Parameter 11: Number of physical qubits allocated to all *T* factories in each module**

The number of active physical qubits across all $n_{T\,\text{factories}}$ factories per module can be computed as $n_{T\,\text{factories}}Q$. After choosing a factory from the T-factory look-up table I, Q becomes the number of physical qubits the factory requires for magic state distillation.

(l) **Parameter 12: Total number of modules in both legs of the macroarchitecture**

The total number of interconnected QPU cryomodules on both legs of the macroarchitecture can be computed as $2n^{\text{per-leg}}_{\text{modules}}$. The parameter $n^{\text{per-leg}}_{\text{modules}}$ is the number of modules per leg or the height of the ladder structure (for details, see Fig. 3).

(m) **Parameter 13: Total number of coherent interconnects in the macroarchitecture**

The total number of width-d coherent interconnects in the ladder macroarchitecture can be computed as $n_{\text{inter-pipes}} \left(3n^{\text{per-leg}}_{\text{modules}} - 2\right)$. FTQC uses these connections to perform intermodule graph-preparation operations on the same-leg or teleporting output to input nodes on the opposite-leg modules (for details, see Fig. 3).

(n) **Parameter 14: Number of allocated physical qubits per module**

The number of allocated physical qubits per module for all components can be computed as $2\left\lceil n_{\text{logical}}/n^{\text{per-leg}}_{\text{modules}} \right\rceil d^2 + n_{T\,\text{factories}}Q + 2l_{\text{transfer-bus}}d^2$.

(o) **Parameter 15: Number of algorithmic logical qubits**, n_{input}

The number of algorithmic logical qubits or width, n_{input}, of the complete input circuit, U. For the time-sliced widgetization we perform on the input algorithm (cf. Fig. 6), all widgets have the same number of input and output nodes, n_{input}.

(p) **Parameter 16: Diamond-norm precision for gate synthesis, ε**

Extensive analytical and numerical studies (see, e.g., Refs. [69,70]) have established that a non-Clifford one-qubit (1Q) rotation gate can be systematically approximated as a chain of Clifford + T, a process known as gate synthesis. Here, the upper bound for the T length scales as $L_\varepsilon = \left\lceil c_0 \log_2(1/\varepsilon) + c_1 \right\rceil$. The diamond-norm precision, ε, indicates the rate at which FTQC utilizes the gate-synthesis decomposition of arbitrary-angle rotations to perform Rz-basis measurements at the end, during each consumption step. We assume the use of the state-of-the-art decomposition approach of *mixed-fallback* [70] by default, which offers $c_0 = 0.57$ and $c_1 = 8.83$, representing an improvement over the previously employed "GridSynth" method [69], which had $c_0 = 3.0$ and $c_1 = 0$.

RRE internally processes non-Clifford angle decompositions, meaning that the constructed graphs do *not* depend on the ε value and can be cached. The diamond-norm precision must be determined accurately and iteratively, following Algorithm 1, which naturally involves solving a logarithmic equation in step 6.

We argue that ε should not be left for the user to set arbitrarily (although RRE allows users to impose a fixed value if needed). Extreme caution is required here because precisions *are*, indeed, equivalent to the fault-tolerant resources required: if ε is chosen too large, the rate condition in step 6 of Algorithm 1 is not met, while if it is too small, the space-time resources will become wasteful and excessively large.

(q) **Parameter 17: Total number of *T* gates, or `T-count`, or N_T^{tot}**

In the graph-state-based estimation method, our *proxy* for the total `T-count`, or N_T^{tot}, is the total number of T-basis measurements required after the arbitrary-angle R_z values are synthesized at the measurement points of the consumption stage. Therefore, we can write `T-count` as $N_T^{\text{tot}} = n^{Rz}_{\text{init}} L_\varepsilon + n^{T}_{\text{init}} = n^{Rz}_{\text{init}} \left\lceil c_0 \log_2(1/\varepsilon) + c_1 \right\rceil + n^{T}_{\text{init}}$.

(r) **Parameter 18: Effective *T* depth**

This parameter estimates the effective T depth for the input logical circuit, U. This can be computed as $\left\lceil n^{T}_{\text{tot}}/\tilde{n}_{\text{logical}} \right\rceil$.

(s) **Parameter 19: Number of *Rz* gates in the input circuit,** n^{init}_{Rz}

We denote the number of non-Clifford-rotation Rz-gates in the complete input algorithm, U, whether written explicitly or contained within other logical gates, as n^{init}_{Rz}. All non-Clifford Rz gates must undergo gate-synthesis decomposition [69,70] to Clifford+T at the end, during the consumption stage.

(t) **Parameter 20: Number of T,$T^\dagger$ gates in the input circuit,** n^{init}_{T}

We denote the number of T and $T^\dagger$ gates in the complete input algorithm, U, whether written explicitly or

contained within other logical gates, as n_T^{init}. We perform this count before performing gate-synthesis Clifford+T decomposition of non-Clifford R_z-gates.

(u) **Parameter 21: Number of Cliffords in the input circuit,** $n_{\text{Clifford}}^{\text{init}}$

The number of explicit Clifford gates in the complete input algorithm, U, which is denoted as $n_{\text{Clifford}}^{\text{init}}$.

(v) **Parameter 22: Number of graph-state nodes,** N

N represents the total number of logical nodes in the complete graph state, $\mathcal{G}$, or the graph size. For widgetized circuits, we calculate a proxy for this quantity from subgraphs as $\tilde{N} = \sum_{i=0,\ldots,n_{\text{widgets}}-1} N_i + \sum_{i=0,\ldots,n_{\text{widgets}}-2} n_i^{\text{outputs}} \approx N$. Here, $n_i^{\text{outputs}} = n_{\text{input}}$ denotes the number of output nodes at step i that must be added to the graph stitched from subgraphs to enable Bell-pair teleportations.

(w) **Parameter 23: Total number of measurement steps in the consumption schedule**

The total number of sequential measurement steps in the consumption schedule over all widgets (repeated or not). In other words, the total number of subsets in the nested sets in the consumption schedule outputted by CABALISER [46]. This quantity governs the run-time of the quantum operation for the graph consumption stage.

(x) **Parameter 24: Total number of measurement steps in the preparation schedule,** $\mathbb{L}^{\text{prep}}$

The total number of sequential measurement steps in the preparation schedule over all widgets (repeated or not). In other words, the total number of subsets in the nested sets in the preparation schedule outputted by Substrate Scheduler [44]. This quantity governs the run-time of the quantum operation for the graph-preparation stage.

(y) **Parameter 25: Total number of widgets or widgetization time steps**, n_{widgets}

The total number of widgets (repeated or not), which is equivalent to the number of time steps in U given the time-sliced decomposition presented in Fig. 6 and Appendix B.

(z) **Parameter 26: Number of distinct widgets**, n_{widgets}^*

n_{widgets}^* is the number of distinct widgets in set $[U_0, \ldots, U_{n_{\text{widgets}}-1}]$ for the time-sliced widgetization of U. This means one can always write $n_{\text{widgets}}^* \leq n_{\text{widgets}}$. This is the number of widgets that FTQC needs to compile.

(aa) **Parameter 27: decoder tock**, $t_{\text{tock}}^{\text{decoder}}$

For our proposed superconducting FT architecture, we assume that it is feasible to engineer generic decoders utilizing modern GPU or FPGA hardware and methods such as MWPM or neural networks. $t_{\text{tock}}^{\text{decoder}}$ is the time required for the decoder to complete decoding d QEC sweeps expressed in seconds. This can be approximated as $t_{\text{tock}}^{\text{decoder}} = dt_{\text{decoder}}$. We assume that it is feasible to engineer generic GPU or FPGA-based decoders with a decoding tock or delay in the same order of magnitude as $t_{\text{tock}}^{\text{decoder}}$ for superconducting architecture. Here, we consider a characteristic decoder cycle time of $t_{\text{decoder}} = 1\ \mu\text{s}$.

(ab) **Parameter 28: Intramodule quantum tock for graph processing,** $t_{\text{tock}}^{\text{intra}}$

The time needed for FTQC to sequentially perform d QEC sweeps for all intramodule quantum operations, which prepare or consume the subgraphs, is computed as $t_{\text{intra}} = 8dt$. Here, t represents the characteristic (dominant) gate time that sets the architectural tock for all intramodular operations. The factor of 8 arises from the fact that in our physical architecture, each surface-code sweep for syndrome extraction circuits necessitates one initialization, two Hadamards, four entangling gates, and one measurement, all of which are assumed to take a duration of t (see, e.g., Ref. [7, Fig. 9]).

(ac) **Parameter 29: Intramodule quantum tock for** ***T*** **factories**

The time required for T factories to distill a T state performing d QEC sweeps sequentially. This can be computed as $8Ct$ for T factories built from rotated surface-code patches.

(ad) **Parameter 30: Total number of available physical qubits**, $n_{\text{avail-phys}}$

Total number of physical qubits available in all $2n_{\text{modules}}^{\text{per-leg}}$ modules of the FTQC.

(ae) **Parameter 31: Number of available logical qubits per module,** $n_{\text{avail-logical}}$

The number of logical qubits available per module, which is allocated to different architectural components and quantum operations. This can be computed as $n_{\text{avail-logical}} = l_{\text{edge}}^2$.

(af) **Parameter 32: Total number of** ***unallocated logical*** **qubits**, $n_{\text{unalloc-logical}}$

Considering all modules and operations of the FTQC, the total number of unallocated *logical* qubits. This parameter can be computed as $2n_{\text{unalloc-logical}}n_{\text{modules}}^{\text{per-leg}}$, where $n_{\text{unalloc-logical}} = l_{\text{edge}}^2 - 2\left\lceil \tilde{n}_{\text{logical}}/n_{\text{modules}}^{\text{per-leg}} \right\rceil - l_{\text{transfer-bus}} - n_{T\,\text{factories}} \left\lceil L_{\text{length}}/\sqrt{2}d \right\rceil \left\lceil L_{\text{width}}/\sqrt{2}d \right\rceil$.

(ag) **Parameter 33: Total number of** ***unallocated physical*** **qubits**

Considering all modules and operations of the FTQC, the total number of unallocated *physical* qubits. This parameter can be computed as $4n_{\text{unalloc-logical}}n_{\text{modules}}^{\text{per-leg}}d^2$.

(ah) **Parameter 34: Total number of** ***allocated logical*** **qubits**

Considering all modules and operations of the FTQC, the total number of allocated *logical* qubits to different architectural components. This parameter can be computed as $2n_{\text{alloc-logical}}n_{\text{modules}}^{\text{per-leg}}$, where $n_{\text{alloc-logical}} = 2\left\lceil \tilde{n}_{\text{logical}}/n_{\text{modules}}^{\text{per-leg}} \right\rceil + l_{\text{transfer-bus}} + n_{T\,\text{factories}} \left\lceil L_{\text{length}}/\sqrt{2}d \right\rceil \left\lceil L_{\text{width}}/\sqrt{2}d \right\rceil$.

(ai) **Parameter 35: Total number of *allocated physical* qubits**

Considering all modules and operations of the FTQC, the total number of allocated *physical* qubits to different architectural components. This parameter can be computed as $4n_{\text{alloc-logical}}n_{\text{modules}}^{\text{per-leg}}d^2$.

(aj) **Parameter 36: Number of concurrent decoding cores at consumption stage**, $\text{cores}_{\text{consump}}^{\text{decoding}}$

In our superconducting architecture, we assume that generic reference classical decoders, equipped with state-of-the-art GPU or FPGA cores, perform decoding cycles. These receive the syndrome and output possible corrective actions, which can, e.g., be performed at the same time as measurements at the ends of consumption steps.

Quantum operations do *not* need to wait for the generic decoders that we have assumed to finish their cycles to perform corrective measurements. One can always consider a staggered type of architecture, where classical and quantum units work together at each step. Moreover, since decoder tocks are typically much longer than quantum tocks, the best strategy is to let as many *concurrent* decoding cores complete their cycles as possible at each consumption step. In this way, the decoding process only adds an overall delay to the consumption stage run-time on top of purely quantum operations. There would be a maximum number of cores that can run concurrently at any consumption step.

We define $\text{cores}_{\text{consump}}^{\text{decoding}}$ as the maximum number of concurrent decoding cores running at any consumption step. Hence, consumption steps often dominate intramodule run-time, so we set $\text{cores}_{\text{consump}}^{\text{decoding}}$ as the available number of concurrent cores for all decoding steps, including graph preparation, teleportation, consumption, or T injection operations. This is calculated as $\text{cores}_{\text{consump}}^{\text{decoding}} = \lceil t_{\text{tock}}^{\text{decoder}}/t_{\text{tock}}^{\text{intra}} \rceil$.

(ak) **Parameter 37: Total QPU area**

This parameter represents the total area occupied by all QPU modules on both legs of the ladder macroarchitecture.

(al) **Parameter 38: Total number of couplers**

This parameter represents the total number of adjustable couplers required for all modules on both legs of the ladder macroarchitecture.

(am) **Parameter 39: Decoding power at consumption stage,** $\text{POW}_{\text{consump}}^{\text{decoding}}$

The decoding power used during the consumption stage is based on a 100-W reference decoding core (aligned with modern GPU or FPGA-based units). Therefore, we can write down $\text{POW}_{\text{consump}}^{\text{decoding}} = 100\text{cores}_{\text{consump}}^{\text{decode}}$ W.

(an) **Parameter 40: Power dissipation at 4-K stage,** $P_{\text{Diss,4 K}}$

Power dissipation for the 4-K stages of all (cryo)modules, in watts.

(ao) **Parameter 41: Power dissipation at 20-mK stage,** $P_{\text{Diss,20 mK}}$

Power dissipation for the 20-mk stages of all (cryo)modules, in watts.

(ap) **Parameter 42: Total graph consumption time,** $t_{\text{consump}}^{\text{tot}}$

The total time to consume subgraphs across all consumption schedule steps is expressed in seconds. $t_{\text{consump}}^{\text{tot}}$ includes the delays required to distill additional T states or prepare the subgraphs, while excluding handover run-time and decoding delays. This is calculated as $t_{\text{consump}}^{\text{tot}} = 8td(L_{\text{prep}}^0 + N_{\text{consump}}^{\text{seq}}) + t_{\text{distill}}^{\text{delay}} + t_{\text{prep}}^{\text{delay}}$, as detailed in Sec. II D 7.

(aq) **Parameter 43: Total intermodular graph handover time,** $t_{\text{handover-inter}}^{\text{tot}}$

The total intermodular time for Bell-state teleportations to hand over (stitch) subgraphs across all scheduling steps. This is the time to teleport the output nodes on the current subgraph on the quantum register on one leg to the input nodes of the subgraph on the other leg for all steps. This is calculated as $t_{\text{handover-inter}}^{\text{tot}} = \sum_{i=0,\ldots,n_{\text{widgets}}-2} t_{\text{handover-inter}}^{i}$ and was detailed in Sec. II D 7.

(ar) **Parameter 44: Overall *T*-distillation delay,** $t_{\text{tot}}^{\text{distill-delay}}$

$t_{\text{tot}}^{\text{distill-delay}}$ is the overall delay included in the total consumption time, $t_{\text{consump}}^{\text{tot}}$, to distill additional T states in the T factories in the same modules to complete consumption-stage measurements (for details, see Sec. II D 7).

(as) **Parameter 45: Overall graph-preparation delay,** $t_{\text{tot}}^{\text{prep-delay}}$

$t_{\text{tot}}^{\text{prep-delay}}$ is the overall delay included in the total consumption time, $t_{\text{consump}}^{\text{tot}}$, to prepare the next subgraph in the schedule. The FTQC pipeline cannot proceed from the current consumption step until the next subgraph is prepared in the next set of modules on the other leg of the ladder (for details, see Sec. II D 7).

(at) **Parameter 46: Overall decoding delay,** $t_{\text{tot}}^{\text{decode-delay}}$

$t_{\text{tot}}^{\text{decode-delay}}$ represents the total decoding delay accumulated from all graph consumption steps, which we must add to the overall FT hardware time. We assume there are *no* decoying delays during subgraph handover operations due to $t_{\text{tock}}^{\text{decoding}} \leq 8t_{\text{inter}}d$. We can determine the total decoding delay, given that $t_{\text{tock}}^{\text{decoding}} > t_{\text{tock}}^{\text{quantum}}$ and $t_{\text{tock}}^{\text{decoding}} > 8Ct$, using $t_{\text{tot}}^{\text{decode-delay}} = \left(8td(L_{\text{prep}}^0 + N_{\text{consump}}^{\text{seq}}) + t_{\text{prep}}^{\text{delay}}\right)\left(t_{\text{tocks}}^{\text{decoding}} - t_{\text{tock}}^{\text{quantum}}\right) + \left\lceil t_{\text{distill}}^{\text{delay}}/8Ct \right\rceil \left(t_{\text{tocks}}^{\text{decoding}} - 8Ct\right)$.

(au) **Parameter 47: Total wall time over a single FT algorithm step,** $t_{\text{hardware}}^{\text{total}}$

$t_{\text{hardware}}^{\text{total}}$ is our estimated *upper bound* for the wall time accumulated from all nonsimultaneous components

running in modules on both legs and coherent interconnects of the architecture of Sec. II C over a single FT algorithm step, U. The algorithmic step should not be confused with n_{widgets} widgetization steps of U. (An algorithmic step occurs when the user requires $n_{\text{algo-reps}}$ repetitions of U on the FTQC.) This wall time includes all inter- and intratocks required at the consumption stages, delays for graph preparation and T distillation, subgraph handover time, and decoding delay. In other words, this is our overall time cost for one step, which can be formulated as $t^{\text{total}}_{\text{hardware}} = t^{\text{total}}_{\text{intra}} + t^{\text{total}}_{\text{handover-inter}} + t^{\text{total}}_{\text{decode-delay}}$ (for details, see Sec. II D 7).

(av) **Parameter 48: Total FT algorithm time,** $t^{\text{total}}_{\text{FT}}$

RRE allows users to specify the number of times, $n_{\text{algo-reps}}$, with which they would like to repeat an exact copy of the complete algorithm U back to back. For these cases, the total FT time is calculated as $t^{\text{total}}_{\text{FT}} = n_{\text{algo-reps}} t^{\text{total}}_{\text{hardware}}$.

(aw) **Parameter 49: Total energy consumption,** E_{tot}

We report estimates for E_{tot}, the *upper-bound* on the total energy usage of the FTQC, considering all modules, quantum and decoding operations, and algorithmic steps, which is unique to RRE.

We identify *three* key power-consuming units that dissipate energy during FT computations. These units include decoding cores with a fixed TDP of $P^{\text{decoding}}_{\text{per-core}} = 100$ W on duty during all QEC operations. We assume a fixed number of concurrent decoding cores equal to $\text{cores}^{\text{decoding}}_{\text{consump}}$ are assigned to all decoding operations, which includes graph preparation, consumption, handover, and T distillation across full $t^{\text{FT}}_{\text{total}}$. This means we can approximate an upper bound to decoding energy consumption per core as $P^{\text{decoding}}_{\text{per-core}} t^{\text{FT}}_{\text{total}}$.

We also include 4-K units with a TDP of $P_{\text{dissip,4 K}}$ and assume a single cooling-efficiency factor of $\eta_{4\,\text{K}} = 500$ for this stage. Additionally, we have lower-temperature MXC (20-mK) units with a TDP of $P_{\text{dissip,20 mK}}$ and assume a single cooling-efficiency factor of $\eta_{20\,\text{mK}} = 10^9$ for this stage. Each unit comprises several power-consuming stages, and we determine these values via thermal analysis of the signal chain (for details, see Sec. II C 3 and [29,68,92]). Overall, we obtain $E_{\text{tot}} = (P^{\text{decoding}}_{\text{per-core}} \text{cores}^{\text{decoding}}_{\text{distill}} + \eta_{4\,\text{K}} P_{\text{dissip,4 K}} + \eta_{20\,\text{mK}} P_{\text{dissip,20 mK}}) t^{\text{FT}}_{\text{total}}$.

APPENDIX B: WIDGETIZATION VIA SUBCIRCUIT DEPENDENCY GRAPH

In Sec. II D 3, we have established that our resource-estimation tooling supports the widgetization of CIRQ circuits. The primary reason for such a widgetization tool is that fully unrolling a large circuit into its single- and two-qubit operations for a utility-scale application can place severe pressure on memory and compute resources. Moreover, for resource estimation using RRE, it is unnecessary and inefficient to revisit, reexamine, and recompile subcircuits that have already been prepared for use by RRE. To understand the widgetization process, a few key concepts about the nature of CIRQ circuits need to be defined.

a. Structure of CIRQ *circuits*

Definitions For this discussion, we will follow the definitions of quantum circuit objects in the CIRQ documentation. As such, we define a quantum gate as "an effect that can be applied to a collection of qubits" (logical qubits for our purpose). A quantum operation is determined to be a pair consisting of a quantum gate and a list of logical qubits that the gate acts on. A moment is a collection of operations, each acting on a disjoint set of logical qubits. If one imagines a time-ordered list of operations, a moment corresponds to a single time "slice" where operations are being carried out in parallel. Finally, a quantum circuit is a time-ordered sequence of operations that can, if desired, be organized into a time-ordered sequence of moments.

Nested circuits In addition to a library of common gates, CIRQ allows users to define custom gates by providing either a matrix specifying the unitary matrix defining the effect of the gate on logical qubits, or by providing instructions on how to create an ordered list of operations from a list of qubits to act on. The latter of the two options can be thought of as instructions for "decomposing" a gate into constituent gates. By providing a method to define gates in terms of other gates, CIRQ naturally provides the infrastructure to create circuits that are a hierarchy of nested subcircuits. We will focus on circuits comprised of this latter type of "decomposable" gates.

Utilizing this nested subcircuit structure, a user may construct a relatively simple-looking circuit that consists of billions, if not trillions, of single- and two-qubit gates when fully unrolled to an alphabet of indivisible operations. This is particularly true where some subcircuits consist of many repetitions of smaller subcircuits. The strategy employed during widgetization is to identify moderately sized recurring subcircuits (called "widgets") by recursively decomposing and analyzing the nested subcircuit hierarchy. An example of a nested circuit structure is shown in Fig. 14.

Division of subcircuits One may define a measure of subcircuit size, such as the number of gates or active logical qubits. Using this measure, one may decide that any particular subcircuit is "too large" to be a "widget", and therefore the subcircuit needs to be divided into smaller constituent subcircuits. We find it helpful to define two distinct methods for dividing a subcircuit into its constituent parts.

The first, and most natural, is to "decompose" the subcircuit into operations by using the decomposition

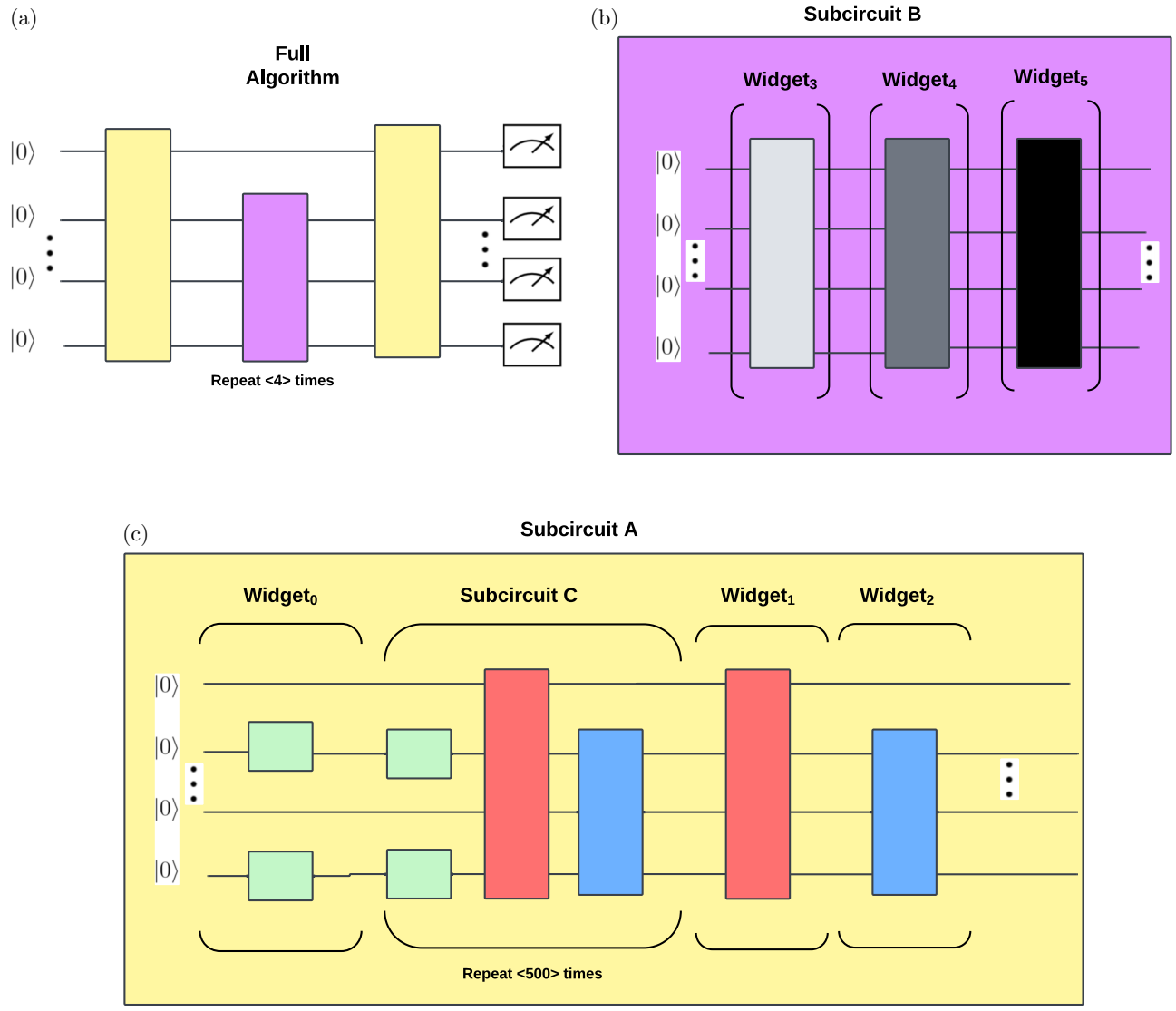

FIG. 14. Widgetization of an example circuit. The full circuit is shown in (a), while its subcircuits and widgets are shown in (b) and (c). (a) An example algorithm consisting of three subcircuits. Subcircuit A (magenta) that is repeated 4 times and subcircuit B (yellow) that occurs before and after the four repetitions of subcircuit B. (b) The splitting of subcircuit B into three smaller subcircuits via slicing. In this example, these slices meet the criterion for not requiring further splitting and, as such, are considered widgets. (c) The division of subcircuit A. Subcircuit A consists of widget_0, 500 repetitions of subcircuit C, then widget_1, and finally widget_2.

instructions provided by the gates of the subcircuit. This allows the subcircuit to be divided into constituent subcircuits, each comprised of a single operation. An example of this type of splitting is shown in Figs. 14(a) and 14(c).

The second type of division is to "slice" the subcircuit into collections of moments. The second method becomes essential when a subcircuit that involves many active logical qubits decomposes into many (thousands or tens of thousands) of operations, each acting on a few qubits. For example, a circuit consisting of thousands of Toffoli gates, each acting on three out of a hundred logical qubits. In these cases, the sheer number of constituent operations slows analysis down and provides subcircuits that are far too small to be considered widgets. By slicing, the subcircuit can be divided into fewer constituent subcircuits of moderate size. An example of this type of division is shown in Fig. 14(b).

Graph-state equivalence To accelerate the widgetization process, the user can provide rules that map subcircuits to unique identifiers, defining classes of "graph-state-equivalent" subcircuits. If two subcircuits are mapped to the same identifier, it is assumed that they will be compiled into graph states that consume equivalent resources. Only one representative of each graph-state equivalence class must be split further and/or compiled. This yields substantial savings in classical processing time when performing both widgetization and resource estimation.

b. Widgetization

To widgetize U without fully unrolling down to single- and two-qubit operations, we perform the following steps:

(1) Construction of a subcircuit dependency graph. This describes the nested structure of the circuit as a directed graph.
(2) Enumeration of widgets and "stitches" indicating when one widget is executed before another.

Each of these steps will be described below.

c. Construction of the subcircuit dependency graph

The first step in the widgetization process is to construct a directed graph (not to be confused with ASG states) that represents the nested subcircuit structure of the original circuit. In this directed graph, the vertices represent subcircuits. A directed edge connects a parent vertex to a child vertex if the child vertex represents the constituent subcircuits of one of its parents. Directed edges between parent and child vertices are *ordered* to execute the subcircuits. This graph is created by recursively splitting subcircuits into smaller ones. This recursive splitting continues until a subcircuit meets specific user-defined criteria. The vertices that represent subcircuits and are not split are called leaf vertices.

An example of a criterion that would prevent a subcircuit from being split is if its number of logical qubits or T gates falls below a user-specified threshold. This recursive splitting is performed in a depth-first manner. Figure 15 shows an example of a graph created from the example circuit described in Fig. 14.

d. Enumeration of widgets and stitches

Once a directed graph representing the nested structure of the original circuit is created, it is used to identify widgets and count how many times they appear during circuit execution. To account for teleportation overheads, we also count the number of times any ordered pair of widgets (called "stitches") would be required during execution.

Widgets are identified as unique leaf nodes in the subcircuit dependency graph. Stitches are identified as ordered pairs of widgets if leaf nodes are written out during a depth-first search of the subcircuit dependency graph while preserving the ordering of the edges connecting parent and child nodes.

From the standpoint of formal language theory, circuits constructed this way can be viewed as variables in a context-free grammar, with the productions corresponding to the gate decomposition rules. At the same time, the subcircuit dependency graph can be regarded as a parse tree. Widgets are viewed as terminals, and stitches are interpreted as ordered pairs of terminals. The number

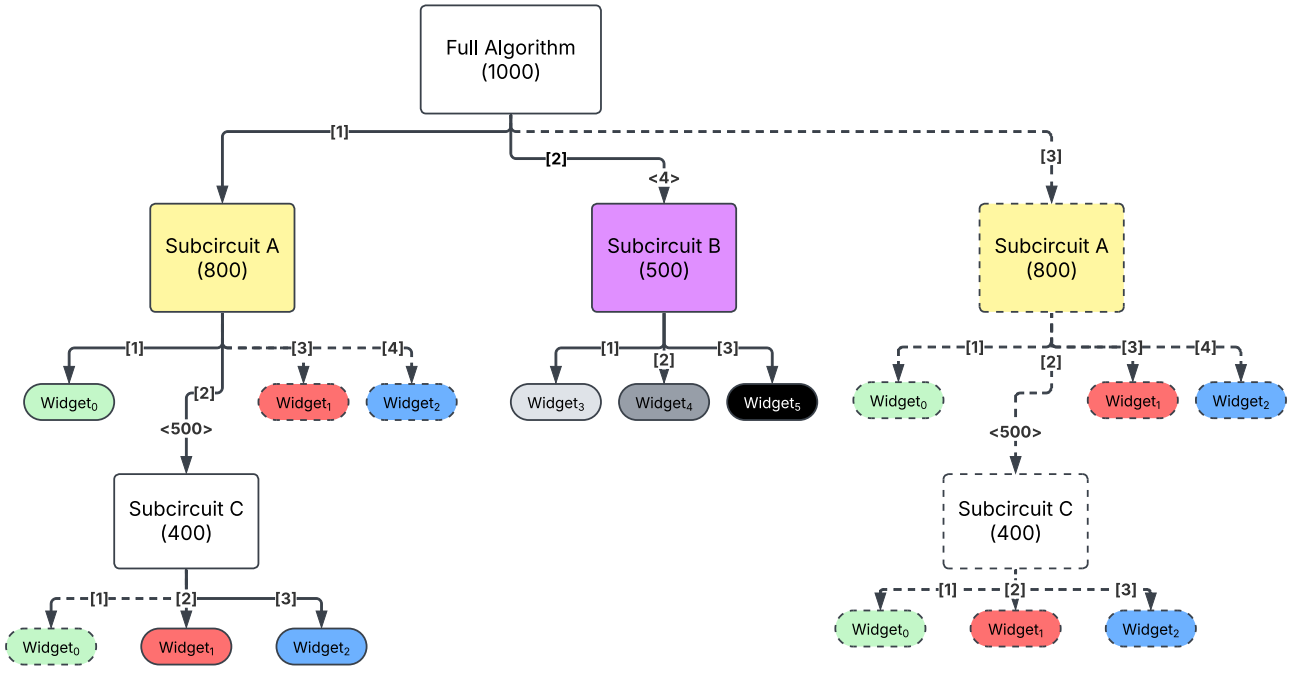

FIG. 15. The subcircuit dependency graph of the example circuit described in Fig. 14. The dashed lines indicate subgraphs that are graph-state equivalent to a previously seen subcircuit and, therefore, are not directly stored in the dependency graph. The quantities in brackets are active qubits in the subcircuit. The numbers in square brackets indicate the ordering of the edges, while the numbers in angle brackets indicate repetitions. An example of a splitting criterion that may have created this graph is to split a subcircuit if its number of active qubits is greater than or equal to 400.

of widgets and stitches is then counted based on their occurrence in the yield of the parse tree.

APPENDIX C: A UNITARINESS VERIFICATION PROTOCOL FOR RRE OUTPUTS

Robust verification of our compilation and estimation methodology across multiple layers was a priority in the design of RRE. To this end, we have developed an exact verification protocol and a series of tests to precisely check the correctness of individual subgraphs, $\{g\}$, and connected schedules, $\{S_{\text{consump}}\}$ and $\{S_{\text{meas}}\}$, as long as the widgets originated from small circuits with a few logical qubits. The protocol includes a step for exact simulation and matrix manipulation in the quantum algorithm, which imposes a widget-size restriction. Recall that every logical qubit involving nontrivial non-Clifford operations can correspond to tens of physical qubits that need exact simulations (one may extend our protocol to tens of logical qubits by performing near-exact tensor-network-based simulations).

This is how our verification protocol works step by step. The input logical unitary, $U = U_{n_{\text{widgets}}-1} \cdots U_1 U_0$, a small algorithm featuring a variety of logical operations (must involve non-Cliffords), is provided to the test module. n_{widgets}, n_{input}, and the number of non-Clifford operations per widget are kept small to keep the test simple and exactly simulatable. Given U, RRE will then run its usual estimation pipeline to generate the sets of

$$\{g_i, S^{g_i}_{\text{frames}}, S^{g_i}_{\text{input}}, S^{g_i}_{\text{output}}, n^{g_i}_{\text{logical}}, S^{g_i}_{\text{consump}}, S^{g_i}_{\text{meas}}, |\text{state}\rangle_i^{\text{CABALISER}}\} \text{ for } i \in \{0, \ldots, n_{\text{widgets}} - 1\}. \quad \text{(C1)}$$

Recall that RRE, by default, initializes from an all-zero state as $|\text{input}\rangle \equiv |s_0 = 0, s_1 = 0, \ldots, s_{n_{\text{input}}-1} = 0\rangle$. Moreover, the RRE output estimation results for U do not play a role in this exact verification protocol. Now, entirely independent of the operations of the compiler, one can construct the exact inverse of the original unitary U for small-size cases through Kronecker matrix multiplications, namely, $U^\dagger$. (As an example, if the widget U_1 is the unitary for a logical `QFT4`, we then need to construct $U_1^\dagger$, i.e., `InverseQFT4`, manually.) If we now apply $U^\dagger$ to the final output state that RRE has generated through CABALISER, $|\text{output}\rangle \equiv |\text{state}\rangle^{\text{CABALISER}}_{n_{\text{widget}}-1}$, by the unitary principle, we should get back the initial state. That is, $U^\dagger |\text{state}\rangle^{\text{CABALISER}}_{n_{\text{widgets}}-1} = |s_0 = 0, s_1 = 0, \ldots, s_{n_{\text{input}}-1} = 0\rangle$ will verify the correctness of graphs and schedules generated by RRE.

In the RRE test modules and procedures, we have simulated and verified the graphs and schedules generated by the software, following the protocol above for selected small input algorithms. These tests have included widgets with non-Clifford instructions such as `QFT3` and `TOFFOLI3`, in addition to various Clifford operations.

APPENDIX D: NOISE MODELING AND LOGICAL ERROR RATE SCALING

Section II C 1 has established that our resource-estimation framework requires a physical-to-logical error scaling law to obtain resource counts. This error scaling law is given by Eq. (1), $p_C = \kappa\,(p/p_{\text{thresh}})^{(d+1)/2}$, where p_C is the logical error rate per cycle, p is the homogeneous physical error rate, and κ and p_{thresh} are the coefficients to be fitted. Below, we describe the computational pipeline we have used to determine the scaling coefficients, as well as the exemplar device-noise model used for the results in this work.

1. Device-noise model

For a preliminary demonstration of RRE, we have chosen a simple noise model that also shows the richness and flexibility the tool offers researchers when considering other devices, circuits, and algorithms. To this end, we have split the noise model into two classes: one simple model for single-qubit gates and another, more realistic, model for two-qubit gates.

For single-qubit gates, we have chosen a depolarizing noise channel, parametrized by the homogeneous physical error rate p. Admittedly, this choice sacrifices realism for ease of implementation, but for an initial baseline demonstration, we have considered it appropriate. The noise model for two-qubit gates has been chosen to be weight-2 Pauli-error channels, with error rates derived from device-specific gate simulations. As mentioned in Sec. II C 1, this paper obtains resource counts for FT execution of circuits protected by a square planar iSWAP surface code. As iSWAP gates were the only two-qubit gate in the surface code, the Pauli error-channel rates have been derived from master equation simulations of parametric iSWAP gates. The simulations have been based on the two-qubit system with one tunable coupler described in Ref. [93]. Process tomography has been carried out to obtain the Pauli error rates from the difference between the ideal iSWAP Pauli transfer matrix and the simulated Pauli transfer matrix.

2. Scaling-coefficient pipeline

To determine the coefficients for the logical-to-physical error rate scaling law, Eq. (1), we have performed a series of surface-code simulations and coefficient fittings by using SINTER [94] and STIM [63].

Given the stochastic nature of the simulations, we have conducted convergence studies to ensure the stability of the coefficients across a large number of shots. This has been done by simulating and fitting coefficients for an increasing number of shots and a maximum number of errors, until the coefficients have converged. The number of shots and maximum number of errors were $2^k \times 10^6$ and $2^k \times 10^5$, respectively, where k ranged from 1 to 11.

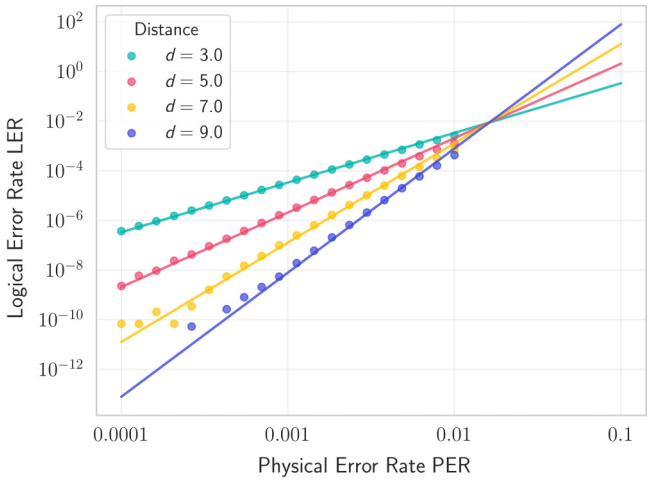

FIG. 16. The logical error rate (LER) to physical error rate (PER) scaling obtained via the scaling-coefficient pipeline in Appendix D 2. The dots are values calculated by the pipeline, and the solid lines are lines of best fit, resulting in the power law given in Eq. (1) with fitting parameters of $\kappa[\text{MWPM}] = 0.009$ and $p_{\text{thresh}}[\text{MWPM}] = 0.016$.

For each fixed number of shots and maximum number of errors, we have swept over 20 values for p ranging from 10^{-2} to 10^{-4} and distances of $d \in \{3, 5, 7, 9\}$. For each p and distance combination, noisy surface-code circuits have been simulated for $10 \times d$ rounds (cycles). Noisy circuits have been constructed by starting with a noiseless iSWAP surface-code circuit written in STIM, provided by Ref. [62]. We have then used the aforementioned noise model to add noise to this circuit in STIM. Syndromes have been decoded using the PyMatching MWPM decoder [64], followed by the calculations of the logical error rates (per cycle). Observed data have then been fitted to the two-coefficient scaling law. The fitted coefficients have been observed to converge to two significant figures by $k = 11$, and the coefficients for $k = 11$ have been used in all downstream resource estimates of Sec. IV. The fit and logical error rates for $k = 11$ are shown in Fig. 16.

[1] C. Gidney, How to factor 2048 bit RSA integers with less than a million noisy qubits, ArXiv:2505.15917.

[2] J. Lee, D. W. Berry, C. Gidney, W. J. Huggins, J. R. McClean, N. Wiebe, and R. Babbush, Even more efficient quantum computations of chemistry through tensor hypercontraction, PRX Quantum **2**, 030305 (2021).

[3] J. J. Goings, A. White, J. Lee, C. S. Tautermann, M. Degroote, C. Gidney, T. Shiozaki, R. Babbush, and N. C. Rubin, Reliably assessing the electronic structure of cytochrome P450 on today's classical computers and tomorrow's quantum computers, Proc. Natl. Acad. Sci. **119**, e2203533119 (2022).

[4] N. P. de Leon, K. M. Itoh, D. Kim, K. K. Mehta, T. E. Northup, H. Paik, B. S. Palmer, N. Samarth, S. Sangtawesin, and D. W. Steuerman, Materials challenges and opportunities for quantum computing hardware, Science **372**, eabb2823 (2021).

[5] M. Field, A. Q. Chen, B. Scharmann, E. A. Sete, F. Oruc, K. Vu, V. Kosenko, J. Y. Mutus, S. Poletto, and A. Bestwick, Modular superconducting qubit architecture with a multi-chip tunable coupler, Phys. Rev. Appl. **21**, 054063 (2024).

[6] R. Acharya *et al.*, (Google Quantum AI and Collaborators), Quantum error correction below the surface code threshold, Nature **638**, 920 (2025).

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

[8] E. T. Campbell, B. M. Terhal, and C. Vuillot, Roads towards fault-tolerant universal quantum computation, Nature **549**, 172 (2017).

[9] D. Litinski, Magic state distillation: Not as costly as you think, Quantum **3**, 205 (2019).

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

[11] P. Webster, M. Vasmer, T. R. Scruby, and S. D. Bartlett, Universal fault-tolerant quantum computing with stabilizer codes, Phys. Rev. Res. **4**, 013092 (2022).

[12] R. Acharya *et al.*, (Google Quantum AI and Collaborators), Suppressing quantum errors by scaling a surface code logical qubit, Nature **614**, 676 (2023).

[13] G. Üstün, A. Morello, and S. Devitt, Single-step parity check gate set for quantum error correction, Quantum Sci. Technol. **9**, 035037 (2024).

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

[15] G. H. Low and I. L. Chuang, Optimal Hamiltonian simulation by quantum signal processing, Phys. Rev. Lett. **118**, 010501 (2017).

[16] L. Lin and Y. Tong, Heisenberg-limited ground-state energy estimation for early fault-tolerant quantum computers, PRX Quantum **3**, 010318 (2022).

[17] A. Paler and A. G. Fowler, OpenSurgery for topological assemblies, ArXiv:1906.07994.

[18] A. Nguyen, G. Watkins, K. Watkins, and V. Seshadri, Lattice-surgery-compiler, https://github.com/latticesurgery-com/lattice-surgery-compiler, (accessed October 13, 2022)

[19] M. E. Beverland, P. Murali, M. Troyer, K. M. Svore, T. Hoefler, V. Kliuchnikov, G. H. Low, M. Soeken, A. Sundaram, and A. Vaschillo, Assessing requirements to scale to practical quantum advantage, ArXiv:2211.07629.

[20] Microsoft azure quantum resource estimation, https://github.com/microsoft/Quantum/tree/main/samples/azure-quantum/resource-estimation (accessed February 17, 2023)

[21] M. McEwen, D. Bacon, and C. Gidney, Relaxing hardware requirements for surface code circuits using time-dynamics, Quantum **7**, 1172 (2023).

[22] D. Forlivesi, L. Valentini, and M. Chiani, Logical error rates of XZZX and rotated quantum surface codes, ArXiv:2312.17057.

[23] A. R. O'Rourke and S. Devitt, Compare the pair: Rotated vs. unrotated surface codes at equal logical error rates, Phys. Rev. Res. **7**, 033074 (2025).

[24] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, Surface code quantum computing by lattice surgery, New J. Phys. **14**, 123011 (2012).
[25] B. J. Brown, K. Laubscher, M. S. Kesselring, and J. R. Wootton, Poking holes and cutting corners to achieve Clifford gates with the surface code, Phys. Rev. X **7**, 021029 (2017).
[26] D. Litinski and F. V. Oppen, Lattice surgery with a twist: Simplifying Clifford gates of surface codes, Quantum **2**, 62 (2018).
[27] A. G. Fowler and C. Gidney, Low overhead quantum computation using lattice surgery, ArXiv:1808.06709.
[28] C. Gidney, N. Shutty, and C. Jones, Magic state cultivation: Growing T states as cheap as CNOT gates, ArXiv:2409.17595.
[29] S. N. Saadatamand, T. L. Wilson, M. J. Hodson, and J. Y. Mutus, Rigetti resource estimation (RRE), https://github.com/rigetti/rigetti-resource-estimation (accessed 6 Feb 2026)
[30] M. Hein, J. Eisert, and H. J. Briegel, Multiparty entanglement in graph states, Phys. Rev. A **69**, 062311 (2004).
[31] S. Anders and H. J. Briegel, Fast simulation of stabilizer circuits using a graph-state representation, Phys. Rev. A **73**, 022334 (2006).
[32] M. K. Vijayan, A. Paler, J. Gavriel, C. R. Myers, P. P. Rohde, and S. J. Devitt, Compilation of algorithm-specific graph states for quantum circuits, Quantum Sci. Technol. **9**, 025005 (2024).
[33] J. Ruh and S. Devitt, Quantum circuit optimisation and MBQC scheduling with a Pauli tracking library, ArXiv:2405.03970.
[34] S. Liu, N. Benchasattabuse, D. Q. Morgan, M. Hajdušek, S. J. Devitt, and R. Van Meter, in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE, Bellevue, WA, USA, 2023), Vol. 1, p. 870.
[35] R. Raussendorf and H. J. Briegel, A one-way quantum computer, Phys. Rev. Lett. **86**, 5188 (2001a).
[36] R. Raussendorf and H. Briegel, Computational model underlying the one-way quantum computer, Quantum Inf. Comput. **6**, 433 (2002).
[37] R. Raussendorf, D. E. Browne, and H. J. Briegel, Measurement-based quantum computation on cluster states, Phys. Rev. A **68**, 022312 (2003).
[38] A. Paler, I. Polian, K. Nemoto, and S. J. Devitt, Fault-tolerant, high-level quantum circuits: Form, compilation and description, Quantum Sci. Technol. **2**, 025003 (2017).
[39] G. Bowen, A. Caesura, S. Devitt, and M. K. Vijayan, Design and efficiency in graph-state computation, ArXiv:2502.18985.
[40] M. A. Nielsen, Cluster-state quantum computation, Rep. Math. Phys. **57**, 147 (2006).
[41] E. Knill, Quantum computing with realistically noisy devices, Nature **434**, 39 (2005).
[42] A. Paler, S. Devitt, K. Nemoto, and I. Polian, Mapping of topological quantum circuits to physical hardware, Sci. Rep. **4**, 4657 (2014).
[43] L. Riesebos, X. Fu, S. Varsamopoulos, C. G. Almudever, and K. Bertels, in *Proceedings of the 54th Annual Design Automation Conference 2017, DAC '17* (Association for Computing Machinery, New York, 2017).
[44] S. Liu, N. Benchasattabuse, and A. Caesura, Substrate Scheduler, `graph-state-generation`, https://github.com/sfc-aqua/gosc-graph-state-generation (accessed 5 March 2023)
[45] M. K. Vijayan, S. Devitt, P. Rohde, A. Paler, C. Mayers, J. Gavriel, M. Stęchły, S. Jones, and A. Caesura, JABALIZER, https://github.com/QSI-BAQS/Jabalizer.jl (accessed November 6, 2022)
[46] A. Robertson, CABALISER, https://github.com/Alan-Robertson/cabaliser (accessed December 6, 2024)
[47] F. Yan, P. Krantz, Y. Sung, M. Kjaergaard, D. L. Campbell, T. P. Orlando, S. Gustavsson, and W. D. Oliver, Tunable coupling scheme for implementing high-fidelity two-qubit gates, Phys. Rev. Appl. **10**, 054062 (2018).
[48] F. Arute *et al.*, Quantum supremacy using a programmable superconducting processor, Nature **574**, 505 (2019).
[49] S. Xu *et al.*, Digital simulation of projective non-Abelian anyons with 68 superconducting qubits, Chin. Phys. Lett. **40**, 060301 (2023).
[50] Y. Wu *et al.*, Strong quantum computational advantage using a superconducting quantum processor, Phys. Rev. Lett. **127**, 180501 (2021).
[51] J. Robledo-Moreno, M. Motta, H. Haas, A. Javadi-Abhari, P. Jurcevic, W. Kirby, S. Martiel, K. Sharma, S. Sharma, T. Shirakawa, I. Sitdikov, R.-Y. Sun, K. J. Sung, M. Takita, M. C. Tran, S. Yunoki, and A. Mezzacapo, Chemistry beyond exact solutions on a quantum-centric supercomputer, Sci. Adv. **11**, eadu9991 (2025).
[52] R. Manenti, E. A. Sete, A. Q. Chen, S. Kulshreshtha, J.-H. Yeh, F. Oruc, A. Bestwick, M. Field, K. Jackson, and S. Poletto, Full control of superconducting qubits with combined on-chip microwave and flux lines, Appl. Phys. Lett. **119**, 144001 (2021).
[53] D. Sank, A. Opremcak, A. Bengtsson, M. Khezri, Z. Chen, O. Naaman, and A. Korotkov, System characterization of dispersive readout in superconducting qubits, Phys. Rev. Appl. **23**, 024055 (2025).
[54] J. L. Mallek, D.-R. W. Yost, D. Rosenberg, J. L. Yoder, G. Calusine, M. Cook, R. Das, A. Day, E. Golden, D. K. Kim, J. Knecht, B. M. Niedzielski, M. Schwartz, A. Sevi, C. Stull, W. Woods, A. J. Kerman, and W. D. Oliver, Fabrication of superconducting through-silicon vias, ArXiv:2103.08536.
[55] D. Rosenberg, D. Kim, R. Das, D. Yost, S. Gustavsson, D. Hover, P. Krantz, A. Melville, L. Racz, G. O. Samach, S. J. Weber, F. Yan, J. L. Yoder, A. J. Kerman, and W. D. Oliver, 3D integrated superconducting qubits, npj Quantum Inf. **3**, 42 (2017).
[56] J. M. Martinis and K. Osborne, Superconducting qubits and the physics of Josephson junctions, ArXiv:cond-mat/0402415.
[57] J. M. Kreikebaum, K. P. O'Brien, A. Morvan, and I. Siddiqi, Improving wafer-scale Josephson junction resistance variation in superconducting quantum coherent circuits, Supercond. Sci. Technol. **33**, 06LT02 (2020).
[58] D. P. Pappas, M. Field, C. Kopas, J. A. Howard, X. Wang, E. Lachman, L. Zhou, J. Oh, K. Yadavalli, E. A. Sete, A. Bestwick, M. J. Kramer, and J. Y. Mutus, Alternating bias

assisted annealing of amorphous oxide tunnel junctions, Commun. Mater. **5**, 150 (2024).
[59] J. V. Damme, S. Massar, R. Acharya, T. Ivanov, D. P. Lozano, Y. Canvel, M. Demarets, D. Vangoidsenhoven, Y. Hermans, J.-G. Lai, V. Rao, M. Mongillo, D. Wan, J. D. Boeck, A. Potocnik, and K. D. Greve, High-coherence superconducting qubits made using industry-standard, advanced semiconductor manufacturing, Nature **634**, 74 (2024).
[60] E. A. Sete, A. Q. Chen, R. Manenti, S. Kulshreshtha, and S. Poletto, Floating tunable coupler for scalable quantum computing architectures, Phys. Rev. Appl. **15**, 064063 (2021).
[61] K. N. Smith, G. S. Ravi, J. M. Baker, and F. T. Chong, Scaling superconducting quantum computers with chiplet architectures, ArXiv:2210.10921.
[62] C. Gidney, MIDOUT, 2023, https://github.com/Strilanc/midout
[63] C. Gidney, STIM: A fast stabilizer circuit simulator, Quantum **5**, 497 (2021).
[64] O. Higgott, PyMatching: A PYTHON package for decoding quantum codes with minimum-weight perfect matching, ACM Trans. Quantum Comput. **3**, 1 (2022).
[65] S. J. Devitt, A. D. Greentree, A. M. Stephens, and R. Van Meter, High-speed quantum networking by ship, Sci. Rep. **6**, 36163 (2016).
[66] J. Chu and F. Yan, Coupler-assisted controlled-phase gate with enhanced adiabaticity, Phys. Rev. Appl. **16**, 054020 (2021).
[67] S. Krinner, S. Storz, P. Kurpiers, P. Magnard, J. Heinsoo, R. Keller, J. Lütolf, C. Eichler, and A. Wallraff, Engineering cryogenic setups for 100-qubit scale superconducting circuit systems, EPJ Quantum Technol. **6**, 2 (2019).
[68] A. Martinez, A. L. Klebaner, J. C. Theilacker, B. D. DeGraff, J. Leibfritz, and J. G. Weisend, in *AIP Conference Proceedings* (AIP, Tucson, Arizona, USA, 2010).
[69] N. J. Ross and P. Selinger, Optimal ancilla-free Clifford + T approximation of z-rotations, Quantum Inf. Comput. **16**, 901 (2016).
[70] V. Kliuchnikov, K. Lauter, R. Minko, A. Paetznick, and C. Petit, Shorter quantum circuits, Quantum **7**, 1208 (2023).
[71] S. Elman, J. Gavriel, and R. Mann, Optimal scheduling of graph states via path decompositions, Phys. Rev. A **111**, 012627 (2025).
[72] Z. Morrell and C. Coffrin, Qc-applications, 2024, https://github.com/lanl-ansi/qc-applications/
[73] A. A. Agrawal, J. Job, T. L. Wilson, S. N. Saadatmand, M. J. Hodson, J. Y. Mutus, A. Caesura, P. D. Johnson, J. E. Elenewski, K. J. Morrell, and A. F. Kemper, Quantifying fault tolerant simulation of strongly correlated systems using the Fermi-Hubbard model, ArXiv:2406.06511.
[74] M. Schneider, J. Ostmeyer, K. Jansen, T. Luu, and C. Urbach, Simulating both parity sectors of the Hubbard model with tensor networks, Phys. Rev. B **104**, 155118 (2021).
[75] M. Qin, H. Shi, and S. Zhang, Benchmark study of the two-dimensional Hubbard model with auxiliary-field quantum Monte Carlo method, Phys. Rev. B **94**, 085103 (2016).
[76] E. W. Huang, R. Sheppard, B. Moritz, and T. P. Devereaux, Strange metallicity in the doped Hubbard model, Science **366**, 987 (2019).
[77] J. Park and E. Khatami, Thermodynamics of the disordered Hubbard model studied via numerical linked-cluster expansions, Phys. Rev. B **104**, 165102 (2021).
[78] X. Wu, Z.-H. Cui, Y. Tong, M. Lindsey, G. K.-L. Chan, and L. Lin, Projected density matrix embedding theory with applications to the two-dimensional Hubbard model, J. Chem. Phys. **151**, 064108 (2019).
[79] G. H. Low and I. L. Chuang, Hamiltonian simulation by qubitization, Quantum **3**, 163 (2019).
[80] J. M. Martyn, Z. M. Rossi, A. K. Tan, and I. L. Chuang, Grand unification of quantum algorithms, PRX Quantum **2**, 040203 (2021).
[81] A. M. Dalzell, S. McArdle, M. Berta, P. Bienias, C.-F. Chen, A. Gilyén, C. T. Hann, M. J. Kastoryano, E. T. Khabiboulline, A. Kubica *et al.*, *Quantum Algorithms: A Survey of Applications and End-to-End Complexities* (Cambridge University Press, Cambridge, United Kingdom, 2025).
[82] K. Obenland, J. Elenewski, K. Morrel, R. S. Neumann, A. Kurlej, J. Belarge, J. Blue, R. Rood, B. Rempfer, and P. Kuklinski, pyLIQTR v1.3.3, https://github.com/isi-usc-edu/pyLIQTR (accessed November 2024)
[83] J. M. Martyn, Y. Liu, Z. E. Chin, and I. L. Chuang, Efficient fully-coherent quantum signal processing algorithms for real-time dynamics simulation, J. Chem. Phys. **158**, 024106 (2023).
[84] S. Zeytinoğlu and S. Sugiura, Error-robust quantum signal processing using Rydberg atoms, ArXiv:2201.04665.
[85] R. Babbush, C. Gidney, D. W. Berry, N. Wiebe, J. McClean, A. Paler, A. Fowler, and H. Neven, Encoding electronic spectra in quantum circuits with linear T complexity, Phys. Rev. X **8**, 041015 (2018).
[86] A. S. Maan and A. Paler, Machine learning message-passing for the scalable decoding of QLDPC codes, ArXiv:2408.07038.
[87] A. Paler and A. G. Fowler, Pipelined correlated minimum weight perfect matching of the surface code, Quantum **7**, 1205 (2023).
[88] S. Varsamopoulos, K. Bertels, and C. G. Almudever, Comparing neural network based decoders for the surface code, IEEE Trans. Comput. **69**, 300 (2020).
[89] https://github.com/ioanamoflic/pandora/.
[90] M. P. Harrigan, T. Khattar, C. Yuan, A. Peduri, N. Yosri, F. D. Malone, R. Babbush, and N. C. Rubin, Expressing and analyzing quantum algorithms with QUALTRAN, ArXiv:2409.04643.
[91] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov, Automated optimization of large quantum circuits with continuous parameters, npj Quantum Inf. **4**, 23 (2018).
[92] N. Raicu, T. Hogan, X. Wu, M. Vahidpour, D. Snow, M. Hollister, and M. Field, Cryogenic thermal modeling of microwave high density signaling, EPJ Quantum Technol. **12**, 124 (2025).
[93] E. A. Sete, V. Tripathi, J. A. Valery, D. Lidar, and J. Y. Mutus, Error budget of a parametric resonance entangling gate with a tunable coupler, Phys. Rev. Appl. **22**, 014059 (2024).
[94] C. Gidney, SINTER, 2023, https://pypi.org/project/sinter/