Evaluation of the classical hardware requirements for large-scale quantum computations

Daan Camps, Ermal Rrapaj, Katherine Klymko, Brian Austin, Nicholas J. Wright
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
Berkeley, CA, USA
{dcamps,ermalrrapaj,kklymko,baustin,njwright}@lbl.gov

***Abstract*—We develop a new model to evaluate the necessary classical computing and networking resources required to support a large-scale fault-tolerant quantum computer based on superconducting qubits and a surface code architecture. We focus specifically on quantum error decoding, which is the main classical computational task required to enable quantum error correction during runtime. Our model reveals that the quantum computer operates at a logical clock speed in the 100–10,000 Hz range, using state-of-the-art quantum error decoders. For a prototypical large-scale quantum chemistry computation, this translates to an overall runtime on the order of months, and this workload is estimated to generate syndrome data for error correction at a rate of 2–500 Gbps depending on whether data compression is used. We estimate the total computational processing power required for online error syndrome decoding equals about 1 petaflop. The results of our analysis show that current computing and networking technology can meet the requirements, in terms of bandwidth, latency, and compute, to support large-scale quantum computation. However, major technological challenges remain both for quantum and classical hardware, including scalable fabrication of high-quality qubits, scalable qubit control, and syndrome communication within a limited power budget.**

***Index Terms*—quantum computation, quantum resource analysis, syndrome decoding, hardware modeling, post-Moore technologies.**

I. Introduction

The slowdown of Moore's law forms a major challenge for the future of high-performance computing. At the same time, it provides new opportunities for innovative compute architectures and novel paradigms to continue to deliver performance gains beyond the exascale era [1].

Quantum computing in particular promises to provide *exponential* performance gains for certain applications of interest to the scientific computing community [2]. It is likely that computational problems that are quantum mechanical in nature, such as problems in materials science [3] and quantum chemistry [4], stand to benefit most from the development of quantum computing technologies. In addition, many physical systems of interest to high-energy physics [5], [6], including finite-density systems [7], chiral gauge theories [8], and baryonic systems [9], require computational resources scaling exponentially with system size in order to overcome sign problems, as complex path integrands cannot be treated as probability distributions for Monte Carlo importance sampling. Simulating four-dimensional lattice gauge theories is crucial for comprehending phase transitions in heavy ion collisions and the early universe, among other phenomena, yet these simulations are currently impractical for classical computing systems.

A number of technological platforms have emerged as leading contenders in the race to build large-scale quantum computers, including superconducting qubits [10], trapped-ion devices [11], and neutral atom hardware [12]. Regardless of the platform, the hardware implementing the qubit states will be imperfect, and quantum error correction (QEC) [13] is essential to protect the quantum information during large-scale quantum computation from the combined effects of decoherence and imperfect quantum gate and measurement operations. This approach is known as fault-tolerant quantum computing (FTQC) [14].

In this work, we evaluate the necessary classical computing and networking resources required to support a fault-tolerant quantum computer during runtime. We particularly focus on the level of compute required for online error decoding, which forms an integral component of FTQC [13]. In order to do so, we assume that the quantum processing unit (QPU) is based on superconducting qubit technology placed in a dilution refrigerator and equipped with a surface code QEC scheme [15], [16], which is one of the most popular approaches for implementing error correction. We focus on superconducting qubits as they operate in the MHz to GHz range and are among the fastest quantum technologies currently being studied and thus pose the most stringent requirements on the classical hardware, including the decoder.

The main contributions of our work are the following:

i. We provide a comprehensive model to evaluate the classical hardware requirements, in terms of network bandwidth, network latency, and processor compute, to support large-scale, universal fault tolerant quantum computations on multiple logical qubits during runtime;
ii. We find that current networking technologies can meet the bandwidth, latency and response requirements to support a quintessentially difficult quantum chemistry computation using a large-scale quantum computer during runtime.

This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Data rates are estimated to be in the gigabit/s to terabit/s range;

iii. We show that online error decoding for quantum problems up to 2,500 logical qubits and 10^9 T-gates (non-Clifford gates that cannot be efficiently simulated classically and that are necessary for creating a universal gate set), is possible with 1 petaflop of computational power using existing error decoders.

The remainder of this article is organized as follows. Section II first provides an overview of related works that model aspects of the quantum computational stack. Next, Section III provides further background about the architectural design of a quantum computer and some concepts from quantum error correction with surface codes that we use throughout our analysis. We develop our model in Section IV: Section IV-A benchmarks and models the data rates generated by a quantum computer during runtime, Section IV-B describes universal quantum computation in a circuit model and estimates the resources required to implement Clifford and non-Clifford quantum operations, Section IV-C models the computational resources required for online error decoding, and Section IV-D concludes the methodology section by integration all previous results in a single resource model. The results obtained with this resource model are reported in Section V. We conclude with a discussion of the results in Section VI.

II. Related work

Quantum resource analysis is concerned with quantifying the computational resources required to solve large, classically intractable computational problems on future large-scale quantum computers. Most analyses provide resources in terms of number of qubits, which corresponds to the space or storage complexity, and the non-Clifford circuit depth, which is the dominant factor in the time complexity. The latter is typically given in terms of the number of T-gates or Toffoli-gates. Noteworthy results include estimates for the chemical simulation of the biologically relevant enzyme cytochrome P450 [17], and for early fault-tolerant simulations of the 2D Hubbard model [18], which is relevant for materials sciences. However, these works focus only on the required *quantum resources*, mostly at the logical encoded level.

Modeling the requirements of quantum computer hardware architectures forms another important research direction. In this category, Beverland et al. [19] provide a quantum resource estimation model that abstracts the layers of the quantum hardware and software stack and argue that qubit technologies should be controllable, fast, and small to scale to practical quantum advantage. Tannu et al. [20] analyzes the quantum instruction bandwidth requirements, while Hoefler et al. [21] model the speed and performance of a fault-tolerant quantum computer and compare these with classical computer hardware. Holmes et al. [22] studies the architectural layout of *magic-state distillation factories*, a way to build high-quality T-states from a larger number of imperfect T-gates and an important component of the QPU, in order to optimize throughput by avoiding routing and congestion issues on the QPU. However, none of these works model the architectural requirements for the classical decoder hardware.

A different area of research is the development and performance analysis of error decoders, both for software and hardware implementations, which are required for online quantum error correction of fault-tolerant quantum computations. Highly optimized graph-based algorithms [23], [24] in combination with parallelization techniques [25]–[27] are used in state-of-the-art software approaches. The performance of decoders has been modeled in recent work [28], [29], but there is still need for an analysis of the resources required to scale online decoding to classically intractable large quantum computations.

In this work, we fill these gaps and develop a novel model to estimate the classical computational resources required to support online quantum error decoding at scale. Our model allows us to estimate the runtime, logical clock speed, decoding latency and bandwidth, and decoding compute intensity for relevant quantum computations.

III. Background

In this section, we first discuss the most important components that make up the quantum computer stack for superconducting qubits, and then provide an overview of the surface code architecture, which is the quantum error correcting code we consider in our paper.

Fig. 1 shows a high-level block diagram of the most important components of the quantum stack we consider. The stack consists of QPU control hardware placed at room temperature, and integrated with a HPC system that handles quantum compilation tasks and online error decoding. This is connected to a dilution refrigerator that contains the QPU kept at near absolute zero (mK) temperatures and additional cryogenic control, error compression and pre-decoding hardware [29]–[31], that can reduce the I/O requirements between room temperature compute and the cryogenic environment to meet stringent cooling power budget constraints [20].

The stack in Fig. 1, from the QPU to compiler, control, and decoder hardware, is assumed to encode and process the quantum information using a specific QEC code, the *surface code* [13], [16], [32]. We choose the surface code as it is (i) well understood (all components required for universal quantum computation on surface code logical qubits have been studied), (ii) it forms a good proxy for many other QEC codes, including color codes [33] and toric codes [34], and (iii) it only requires local qubit connectivity in a 2D planar topology. This 2D connectivity is less challenging to realize experimentally than codes that require long-range connectivity, such as quantum low-depth parity check (qLDPC) codes [35]–[39]. In demonstration of this, a recent experimental breakthrough showed a logical qubit encoded with a distance-3 and 5 surface code on a superconducting QPU demonstrated error suppression [40].

The blueprint for a distance-7 surface code logical qubit is shown in Fig. 2. The code distance is denoted as d and corresponds to the smallest chain of errors on physical qubits

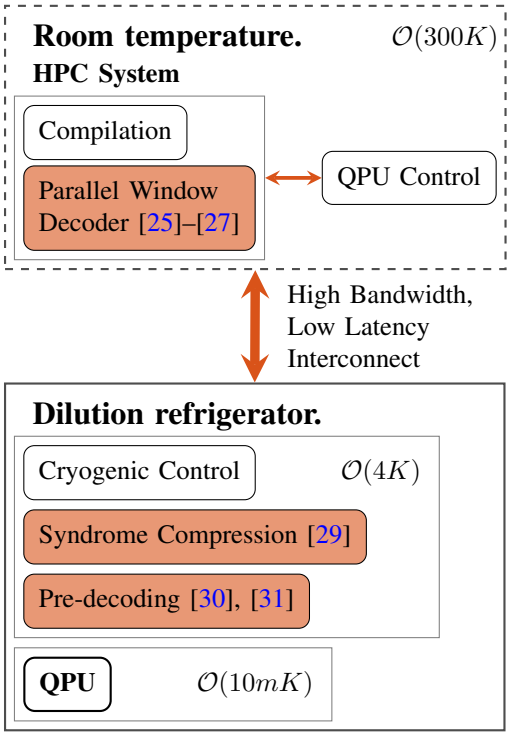

Fig. 1. Block diagram of a quantum computer architecture based on superconducting qubits: classical control hardware at room temperature is interfaced through a high bandwidth, low latency interconnect with different layers in a cryogenic environment, including control, QEC, and the QPU. The QPU and controllers outside the dilution refrigerator are interconnected with an HPC system that compiles the quantum program and supports the online decoding using a parallel window decoder. In this work, we focus on the colored components in the stack to model error decoding at scale.

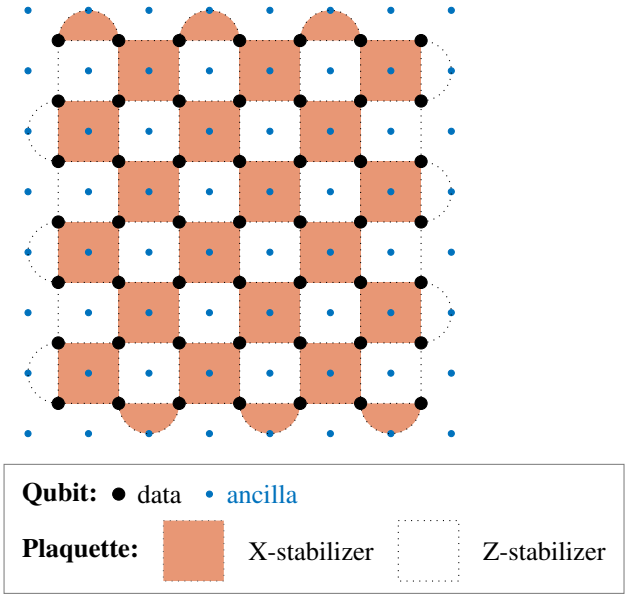

Fig. 2. Distance $d = 7$ rotated surface code logical qubit $|\psi\rangle$ consisting of d^2 data qubits (black dots), ancilla qubits (blue dots), X-stabilizers (colored plaquettes), and Z-stabilizers (white plaquettes) [41].

that cannot be protected. The logical qubit is made up of two nested square lattices of physical qubits that are respectively the d^2 *data qubits* (black dots), which store the logical quantum information, and $d^2 - 1$ *ancilla qubits* (blue dots), which are used to perform stabilizer measurements through parity checks. Each ancilla qubit that is used, is part of a *plaquette* (dashed lines in Fig. 2) and is connected to the data qubits located on the vertices of the plaquette. The plaquettes come in two different kinds as indicated by their white and blue colors in Fig. 2, and measure respectively Z- and X-parity checks to detect phase- and bit-flip errors. These are also known as the Z- and X-stabilizer measurements and project the encoded logical qubit back into a valid code subspace, also known as the *Pauli frame*. The stabilizer measurements are constructed in a way to not measure the encoded logical qubit state (and therefore destroy the quantum information through state collapse), only information about the *error syndrome* is extracted. A surprising result in QEC is that it suffices to protect against phase- and bit-flip errors even though physical quantum errors happen in a continuum [2].

The error syndrome, i.e., the parity check measurements, can be analyzed to determine the Pauli frame of the logical qubit. This process is known as error syndrome *decoding* and is the main computational problem that the classical support hardware in Fig. 1 has to solve. Provided that every physical error mechanism that appears in the hardware flips at most two parity check detectors, the error syndrome maps to a graph-like error model. This assumption agrees well with common noise models assumed for surface codes and we adopt it for the remainder of this paper as an underlying premise. Graph-like error syndromes can be analyzed as minimum-weight perfect matching (MWPM) problems [16], and can be solved using a variant of the blossom algorithm [42]. Naive implementations of the blossom algorithm scale quadratically in the number of vertices of the underlying graph; recent optimized implementations in Sparse Blossom [23] and Fusion Blossom [24] achieve a scaling that is closer to linear. In Sections IV and V, we benchmark and model MWPM decoders to estimate the compute intensity required to support large-scale FTQC.

The logical qubit in Fig. 2 encodes the quantum information in d^2 data qubits and the ancilla qubits are used to measure $d^2 - 1$ stabilizers, which leaves *one* logical degree of freedom $|\psi\rangle$. This redundant encoding of the logical information leads to an exponential suppression of the error provided that the physical error rates in the system are low enough. A logical error only occurs if there is a chain of single-qubit physical errors in the lattice that is equivalent to a logical operator, typically a chain of errors that crosses the lattice either horizontally (logical-Z error) or vertically (logical-X error). The minimal length of such an error chain is d. These type of events are undetectable and lead to a code failure. When the physical error rate rate is small enough, increasing d reduces the likelihood of an error chain spreading across the lattice, leading to an exponential suppression of the logical error. A common heuristic for the *logical error rate* e_ℓ of a surface code qubit that we use as a model throughout our analysis

is [43],

$$e_\ell(e_\mathrm{p}, d) = 0.1\,(100 \cdot e_\mathrm{p})^{(d+1)/2}, \tag{1}$$

where e_p is the physical error rate in the system and d the code distance.

IV. Methods

This section describes our methodology to model the data generation rate, run times, and computational complexity of error decoding.

A. Syndrome data generation rates

The effectiveness of the error suppression in Eq. (1) relies on continuously repeated measurement of the X- and Z-stabilizers in order to remove entropy from the data qubits and avoid accumulation of physical errors. The whole procedure of syndrome extraction is expected to operate in the MHz range for superconducting qubits. We will assume throughout our analysis that each *code cycle*, or round of syndrome extraction, takes $t_\mathrm{cycle} = 1\,\mu s$ [10].

Each code cycle generates $\mathcal{O}(d^2)$ raw bits of information and the raw data generation thus scales quadratically in the code distance d with a prefactor of $t_\mathrm{cycle}^{-1} \propto \mathcal{O}(10^6)$. Fig. 3 illustrates the quadratic scaling of the data generation rates for a single logical qubit encoded in a rotated surface code simulated with Stim [44], a fast simulator for quantum stabilizer circuits. Data is shown for the complete syndrome ($\circ$-marks, blue), as well as the rate of non-trivial, nonzero events at three different physical error rates of 10^{-2} ($+$-marks), 10^{-3} ($\triangle$-marks) and 10^{-4} ($\times$-marks). We observe that the data rate of the non-trivial detection events exhibits a linear dependence on the physical error rates e_p. This allows for syndrome compression schemes to reduce the bandwidth requirements [29]. We capture this dependence by modeling the data rate (in bps) by,

$$R_\ell(e_\mathrm{p}, d) = \begin{cases} d^2 \cdot t_\mathrm{cycle}^{-1}, \\ 4 \cdot e_\mathrm{p} \cdot d^2 \cdot t_\mathrm{cycle}^{-1}, \end{cases} \tag{2}$$

where the first formula is the uncompressed syndrome data rate, and the second formula gives the rate of nonzero events. These models are shown with full and dotted black lines in Fig. 3.

The main takeaway from Fig. 3 is that a single logical qubit generates Mbps to Gbps of uncompressed syndrome data, and kbps to Mbps of non-zero events.

B. Modeling universal fault-tolerant quantum computation

In this section, we introduce our model of universal quantum computation in quantum computer based on a surface code. We will use this model to estimate the requirements on the classical support hardware.

The gate set (or *instruction* set) for universal quantum computation that we make use of is known as the Clifford+T gates, and consists of the H-, S-, CX-, and T-gates. Every

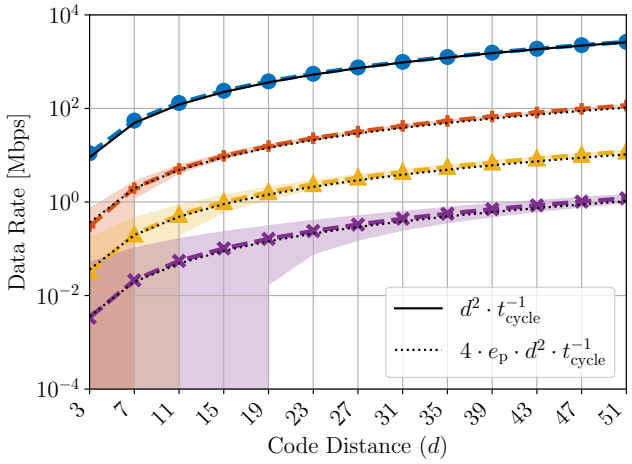

Fig. 3. Raw syndrome ($\circ$, blue) and compressed detector graph data generation rates as a function of code distance d for a rotated surface code quantum memory shown in Fig. 2. The raw data generation rates are shown in dashed lines, and the number of non-trivial events are shown in the dotted lines for physical error rates of 10^{-2} ($+$, red), 10^{-3} ($\triangle$, orange), and 10^{-4} ($\times$, purple). One sigma intervals are shown as shaded bars for 10,000 shots. The models of Eq. (2) are shown in black. Data generated with Stim [44].

logical circuit on Q_ℓ qubits can be written in the Clifford+T gate set as,

$$\prod_{j=1}^{Q_\ell} \prod_{i=1}^{D} H\, S^{a_i}\, T\, (\mathrm{CX}_{j,c(j,i)})^{b_i}, \tag{3}$$

where D is the circuit depth, the parameters $a_i, b_i \in \{0, 1\}$, and $c(j, i)$ returns the control (or target) qubit for the ith CX gate on the jth qubit. The total number of T-gates in Eq. (3) is $n_\mathrm{T} = Q_\ell \cdot D$. Logical measurements are not included explicitly in Eq. (3), but are assumed to be supported both mid-circuit during runtime and at the end.

The implementation of the Clifford gates (H, S, CX), which can be efficiently simulated classically [45], and the logical qubit measurements require significantly fewer resources than implementing non-Clifford T-gates. The latter requires the availability of resource states known as magic states, *online error decoding*, and feed forward. We discuss both the implementation of the Clifford gates and non-Clifford gate in more detail over the next two sections.

1) Complexity of Clifford gates: The single-qubit Clifford H- and S-gates each require $\mathcal{O}(2 \cdot d)$ surface code cycles to implement. A logical measurement in the Z-basis requires $\mathcal{O}(d)$ cycles. The two-qubit CX Clifford gate is more difficult to implement in a surface code quantum computer than a single-qubit Clifford gate because the logical qubits (Fig. 2) are typically expected to be laid out in a 2D grid on the QPU and can only interact along their edges, see also Fig. 6. The standard algorithm for a CX-gate that maps to a 2D grid is based on a procedure known as *lattice surgery* [41]. The process is summarized in Fig. 4. It first prepares an ancilla in the $|+\rangle$-state, which is merged with the control qubit $|c\rangle$ using

a $Z \otimes Z$-parity check measurement (m_{ZZ}), requiring $\mathcal{O}(2 \cdot d)$ cycles of error correction. Next, the ancilla is merged with the target qubit $|t\rangle$ using an $X \otimes X$-parity check measurement (m_{XX}) and then split again. This again requires $\mathcal{O}(2 \cdot d)$ error correction cycles. Finally, the ancilla is measured in the computational basis (m_+) using another $\mathcal{O}(d)$ cycles.

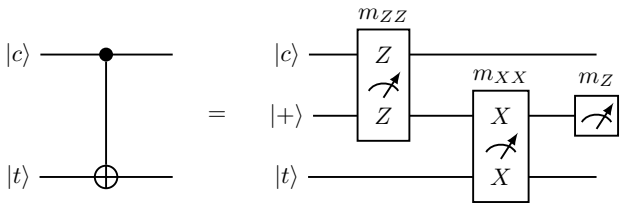

Fig. 4. Circuit to implement a CX-gate between control $|c\rangle$ and target $|t\rangle$ using lattice surgery and an ancilla qubit initialized in the $|+\rangle$ state [41], [46].

The complete CX-gate using lattice surgery thus requires at most $\mathcal{O}(5 \cdot d)$ code cycles. At the end of the circuit in Fig. 4, the Pauli frame of $|c\rangle$ is updated with a phase flip if $m_{XX} \neq 0$, while the Pauli frame of $|t\rangle$ is updated with a bit flip if $m_{ZZ} \oplus m_Z \neq 0$. Clifford gates map Pauli frames onto Pauli frames and thus these Clifford corrections do not have to be implemented on the QPU, but can be tracked classically. If the circuit consists of only Clifford operations, this correction be done offline after the full circuit has been executed. As we will see in the next section, this is no longer possible when non-Clifford operations, i.e. T gates, are included in the circuit.

In conclusion, the time complexity of an HSCX-sequence with logical measurement scales as $\mathcal{O}(10 \cdot d)$ code cycles, i.e.,

$$\Delta_{\text{HSCX}} = 10 \cdot d \cdot t_{\text{cycle}}. \tag{4}$$

2) Complexity of T-gates: The fault-tolerant implementation of the T-gates in the circuit of Eq. (3) forms the main challenge and potential bottleneck of implementing logical operations. In this section, we analyze the steps required to do so and model their time complexity.

T-gates are typically implemented using a two-step process. First, a high-fidelity $|T\rangle$-state is prepared from a collection of noisy, low-fidelity $|T\rangle$-states through a process known as *magic state distillation* (MSD) [47]. The time complexity of MSD scales as,

$$t_{\text{MSD}} = t_{\text{dist}} \cdot d \cdot t_{\text{cycle}}, \tag{5}$$

where t_{dist} is the relevant distillation timescale which depends on the specific configuration of the magic state factory. Throughout this paper, we use the fast surface code configuration introduced in [32]. This configuration uses 15-to-1 state distillation factories that produce one magic state $|T\rangle$ every d code cycles, i.e., $t_{\text{dist}} = 1$. Their implementation requires 132 surface code *patches*, where a patch corresponds to a distance-d logical qubit (cfr. Fig. 2).

Next, the $|T\rangle$-state has to be routed to and *injected* into the logical data qubit $|\psi\rangle$ in order to apply the the T-gate to the logical qubit state. This is achieved using the circuit shown in Fig. 5 which includes a logical measurement of the ancilla carrying the $|T\rangle$-state and a feed-forward, corrective S-gate that depends on the result of the logical measurement. The S-gate has to be explicitly implemented on the QPU before proceeding with the computation and in order to do so, we need to decode the historical error syndrome data in order to know the result of the logical measurements. In other words, the application of the T-gate poses a stop to the subsequent operations on a particular data qubit until the measurement outcome is resolved.

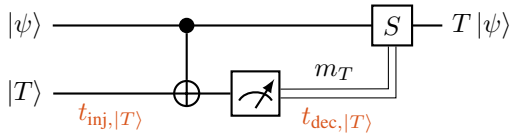

Fig. 5. Circuit for $|T\rangle$-state injection highlighting the two main causes for a delay in our model of quantum computation: the time $t_{\text{inj},|T\rangle}$ it takes to route the $|T\rangle$-state to the data qubit $|\psi\rangle$ and inject it using the CX-gate, and the time $t_{\text{dec},|T\rangle}$ it takes to decode the syndrome of the $|T\rangle$-state to decide if the S-gate has to be applied to the logical qubit $|\psi\rangle$.

We estimate the routing time $t_{\text{route},|T\rangle}$ of the $|T\rangle$-state by modeling the design of the QPU, for instance using the fast surface code QPU layout [32] which is depicted in simplified format in Fig. 6. Each square patch in Fig. 6 corresponds to a $d \times d$ surface code qubit and the color of the square indicates its purpose: yellow patches are used for distilling and storing the $|T\rangle$-states, gray patches are ancillary patches used for lattice surgery, and purple patches are the data qubits in the processor. The number of patches in the QPU scales as [32],

$$n_{\text{patch}} = 2Q_\ell + \sqrt{8Q_\ell} + 132. \tag{6}$$

The number of physical qubits in the QPU scales as,

$$Q_{\text{p}} = 2 \cdot d^2 \cdot n_{\text{patch}}, \tag{7}$$

which includes both the data qubits and ancilla qubits that make up the logical qubit in Fig. 2.

The number of steps required to route the $|T\rangle$-state from the distillation zone to a data qubit scales on average as the radius of the QPU, $\mathcal{O}(\sqrt{n_{\text{patch}}})$. Each step uses patch deformations that take $\mathcal{O}(d)$ code cycles. Throughout our analysis, we however assume that the $|T\rangle$-state routing does not congest the QPU [22] network. As long as this condition is satisfied, the true routing time can be hidden during the computation and does not form a bottleneck. In our model, we allow for a constant effective routing time of $t_{\text{route},|T\rangle} = 2 \cdot d \cdot t_{\text{cycle}}$ time, independent of the QPU size. The $|T\rangle$-state is assumed to be available at the data qubit within this effective routing time. The injection time combines the effective routing time with the application of the logical CX-gate and measurement in Fig. 5 and scales as,

$$t_{\text{inj},|T\rangle} = 7 \cdot d \cdot t_{\text{cycle}}. \tag{8}$$

In the next section we discuss how to estimate the decoding time, $t_{\text{dec},|T\rangle}$.

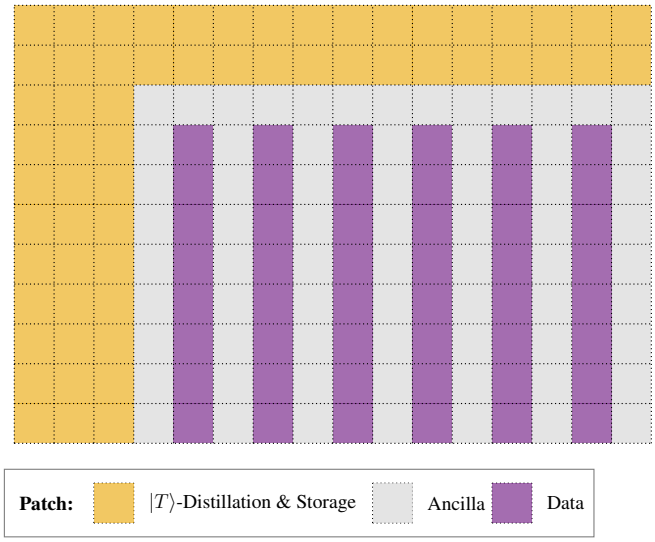

Fig. 6. Simplified quantum processor layout based on the fast surface code configuration of [32]. The processor is partitioned into patches that handle magic state distillation, ancillary patches, and patches storing the data qubits. Each patch is a distance-d surface code logical qubit.

C. Modeling the decoder

In this section, we start by motivating the need for online decoding of T-gates, discuss the potential backlog problem this may cause, and review (parallel) window decoding as a way to overcome this. We then benchmark the performance of an (interrupted) MWPM decoder and use the previous results to develop a model to estimate the overall decoding time $t_{\text{dec},|T\rangle}$ and computational complexity.

1) Online decoding with a parallel window decoder: We need a decoder that is both fast and accurate enough to reliably process the feed forward step required in the T-gate circuit in a timely manner. The speed of the decoder is important for two reasons: (i) we do not want to drastically slow down the logical clock speed at which the circuit in Equation (3) is executed so we can retain the theoretical quantum speedups, and (ii) we need to avoid the *backlog* problem [13] that can grind the computation to a halt. Sufficiently high decoding accuracy is desirable as to not increase the logical error rate Eq. (1) by too much. Online, or real-time, decoding is a syndrome decoder that meets these stringent requirements and is necessary for useful quantum computations.

We remark that for the state injection circuit in Fig. 5, it suffices to have confidence in the m_T-measurement in order to apply the S-gate even in the presence of a delay $t_{\text{dec},|T\rangle}$. The Pauli frame of the data qubit $|\psi\rangle$ might have changed during the delay, but this does not affect the application of the S-gate and can be decoded in the next cycle.

A decoder that can realize a decoding time $t_{\text{dec},|T\rangle}$ that does not (significantly) grow with the amount of syndrome data to be processed is essential for online decoding. Since the time complexity of MWPM grows at best near-linear in the graph size, this type of scaling cannot be realized by a MWPM decoder processing the full syndrome history, which is growing linearly with time.

Sliding window decoding [16] mitigates this issue by: (i) splitting up the syndrome history into overlapping windows of size d, (ii) decoding the syndrome data inside the first window with an inner decoding algorithm (for example MWPM), (iii) committing the Pauli frame corrections in the first half of the window, i.e., the first $(d+1)/2$ cycles, the other half of the window is the buffer zone, and (iv) sliding the window forward in time by $(d+1)/2$ cycles to repeat the process for the next window. The sliding window decoder processes $\mathcal{O}(d^3)$ data at each step and the computational complexity per step thus does not necessarily grow with time. It is however still essential for the sliding window decoder to not be too slow: as long as the decoder can process $\mathcal{O}(d^3)$ syndrome data within one code cycle the pace of the sliding window will keep up in real-time with the syndrome generation rate. Since the window size and thus the decoding time complexity still grows with d, there exists a value for d beyond which the sliding window decoder cannot keep up with this rate.

Parallel window decoding [25]–[27] is a recent extension of sliding window decoding that can tolerate an inner decoder that processes syndrome data in theory at an arbitrary rate and as such fully overcomes the backlog problem. The parallel window algorithm is illustrated for distance $d = 5$ in Fig. 7. It consists of two steps. First, the syndrome data is partitioned in non-overlapping windows of size $3d$ code cycles, with d code cycles separating the windowns (round A). All windows can be decoded completely independently using MWPM. The outer d cycles on the left and right serve as a buffer zone and only the results for the middle d cycles are committed. Next, in round B, the windows of size $3d$, which lie in between the committed zones from round A, are decoded with MWPM and stitched together with the committed windows from round A to decode the full history. The number of windows in both round A and B scales as,

$$n_{\text{window}}(t) = \left\lceil \left\lceil \frac{t}{t_{\text{cycle}}} \right\rceil \Big/ 4d \right\rceil = \left\lceil \frac{n_{\text{cycle}}(t)}{4d} \right\rceil . \tag{9}$$

If we assume the availability of sufficient computational resources, perfect parallel scaling, and ignore the communication overhead, then parallel window decoding can process any amount of syndrome history in a time complexity independent of the amount of syndrome data. Under these conditions, we can achieve online decoding.

We remark that both the sliding and parallel window approaches can match the accuracy of a decoder with access to the full syndrome history. The probability of a syndrome measurement outside the buffer zone impacting the results in the commit zones decreases exponentially with the size of the buffer zone. The setup in Fig. 7 achieves high decoding accuracy in practice [26].

Fig. 8 reports timings for PyMatching to decode windows of syndrome data of size $3d$. The syndrome data is generated by Stim assuming a physical error rate of 10^{-3}. Timings are

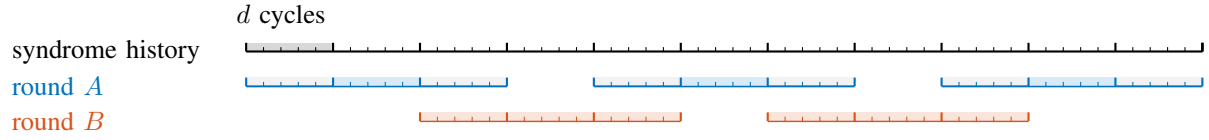

Fig. 7. Setup for parallel window decoding for code distance $d = 5$ in two parallelizable decoding rounds. The middle d cycles of the blue windows are committed in round A, with the outer d cycles on the left and right acting as a buffer zone. In round B the intermediate windows are decoded and stitched together with the committed windows of round A to recover the full syndrome history [26].

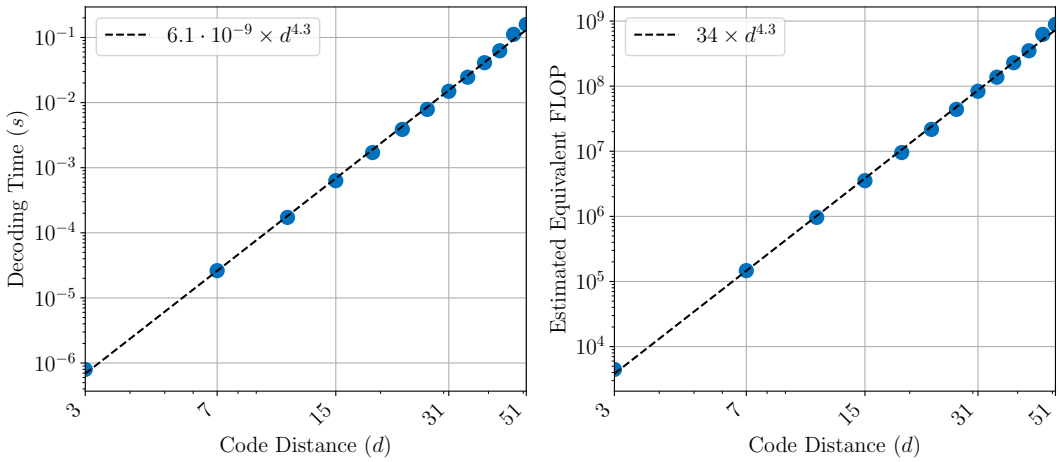

Fig. 8. Results from benchmarking the MWPM decoder as implemented in PyMatching [48] on an AMD EPYC 7763 CPU. **Left:** Decoding time per window of size $3d$ in function of code distance d. Error syndromes are generated by Stim [44] for a physical error rate of 10^{-3}. Timings are averaged over 500 windows. **Right:** Estimated equivalent operations to decode a window of size $3d$ as a function of code distance d.

shown on the left panel and averaged over 500 shots. We observe that the average time to decode a window scales as,

$$t_{\text{window}}(d) = 6.1 \cdot 10^{-9} \times d^{4.3}. \tag{10}$$

The right panel of Fig. 8 reports the *estimated equivalent operations* (FLOP$^{\text{EE}}$) computed by assuming that PyMatching achieves 10% of peak performance, i.e.,

$$\text{FLOP}^{\text{EE}}_{\text{window}}(d) = 0.1 \cdot \mathcal{P}_{\text{peak}} \cdot t_{\text{window}}(d) = 34 \times d^{4.3}, \tag{11}$$

using $\mathcal{P}_{\text{peak}} = 3.58\,\text{TFLOP}/s$ for the 64 core AMD EPYC 7763 CPU.

2) Probabilistic model for interrupted inner decoder: The decoding time depends on the specific realization of the syndrome. To capture this behavior and extend the model of Eq. (10), we adapt the probabilistic model introduced in [28]. We review and extend this model next.

Let Ω denote the sample space of all possible error syndromes. The decoding time then is a discrete random variable $X_{\text{dec}} : \Omega \to \mathbb{N}$, where for each syndrome $s \in \Omega$, $X_{\text{dec}}(s)$ is the decoding time $k \in \mathbb{N}$, given as number of surface code cycles t_{cycle}. We denote $\Pr_{\text{dec}}(k) = \Pr(X_{\text{dec}} = k)$ as the probability that the decoding takes k cycles, and use a simple binomial distribution as a model for the decoding time,

$$\Pr_{\text{dec}}(k) = \binom{N}{k} p^k (1-p)^{N-k}, \tag{12}$$

where $N \in \mathbb{N}$, $p \in [0, 1]$, and $k \in \{0, \ldots, N\}$. The maximum number of cycles to decode any syndrome in Ω is $k^{\max}_{\text{dec}} = N$ and the average decoding time is given by $\mathbb{E}(X_{\text{dec}}) = Np$.

In order to find appropriate parameters N, p for this model, we run another benchmark, reported in Fig. 9, to measure the distribution of the decoding time as a function of code distance. We model the data shown in red in Fig. 9 with Eq. (12) and the parameters,

$$N = 6.1 \cdot d^{4.3}, \quad p = e_{\text{p}}, \tag{13}$$

where $e_{\text{p}} = 10^{-3}$. The average runtime of this probabilistic model matches Eq. (10), while the worst case complexity scales as $k^{\max}_{\text{dec}} = 6.1 \cdot d^{4.3}$.

Since the decoding time is upper bounded by the worst case time complexity, we allow the decoder to stop prematurely after some time $M \leq k^{\max}_{\text{dec}}$. The runtime distribution of an interrupted decoder is given by [28], [29],

$$\Pr^{[M]}_{\text{dec}}(k) = \begin{cases} 0 & \text{if } k > M, \\ \frac{\Pr_{\text{dec}}(k)}{\Pr_{\text{dec}}(k \leq M)} & \text{otherwise.} \end{cases} \tag{14}$$

An interrupted decoder fails whenever it encounters a syndrome s for which the decoding time is longer than M. We assume that the logical error rate for an interrupted decoder is equal to the weighted linear combination of the logical and

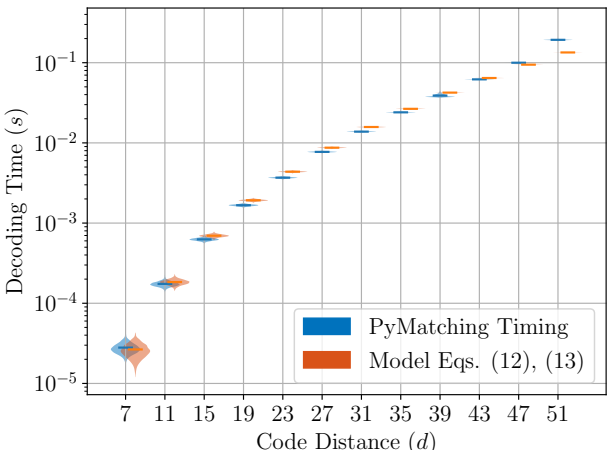

Fig. 9. Distributions of runtimes of the PyMatching decoder on windows of size $3d$ as a function of code distance d (*blue*). Results are shown for a physical error rate of 10^{-3} and 2,500 shots. The probabilistic models of the decoder obtained from Eqs. (12) and (13) are shown in *red*.

physical error rates with weights respectively given by the decoding success and failure probabilities, i.e.,

$$e_\ell^{[M]} = e_\ell \cdot \Pr_{\text{dec}}(k \leq M) + e_{\text{p}} \cdot \Pr_{\text{dec}}(k > M). \tag{15}$$

We remark that this is a different criteria than assumed in [28].

We can now summarize our analysis by the following estimate of the decoding time,

$$t_{\text{dec},|T\rangle} = 2 \cdot M \cdot t_{\text{cycle}}, \tag{16}$$

where the factor two originates from the two rounds of parallel window decoding, the factor $M \cdot t_{\text{cycle}}$ is the maximum decoding time in seconds, and we use $t_{\text{cycle}} = 1\mu s$.

D. Complete model

In this section, we formulate the complete model that we use to estimate the computational resources required for a large-scale quantum computation.

The key input parameters for the model are: (i) the number of logical qubits (Q_ℓ), (ii) the number of T-gates (n_{T}) required for the computation, and (iii) the error budget ε which is the probability that the computation is succesful.

Based on this, we minimize the runtime of the application by minimizing the decoder interruption time M and surface code distance d. The runtime can then be used to estimate the equivalent operations required for the decoder.

1) Optimizing application runtime: The application runtime of the circuit in Eq. (3) is equal to,

$$t_{\text{run}} = n_{\text{T}} \cdot (\Delta_{\text{HSCX}} + \Delta_T) = n_{\text{T}} \cdot \Delta_{\text{HSTCX}}, \tag{17}$$

where Δ_{HSCX} is given in Eq. (4) and Δ_T is the runtime for a single T-gate. Δ_T is estimated as the sum of Eqs. (5), (8) and (16) and the network latency ℓ, i.e.,

$$\begin{aligned} \Delta_T &= t_{\text{MSD}} + t_{\text{inj},|T\rangle} + t_{\text{dec},|T\rangle} + \ell, \\ &= (8d + 2M) \cdot t_{\text{cycle}} + \ell. \end{aligned} \tag{18}$$

We introduced the network latency to model the delay in transmitting the syndrome from the QPU to the decoder and transmitting the decision about the S-gate back to the QPU. Plugging this into Eq. (17), we get that the application runtime t_{run} scales as

$$t_{\text{run}} = n_{\text{T}} \cdot ((18d + 2M) \cdot t_{\text{cycle}} + \ell)\,. \tag{19}$$

We need to choose a code distance d and interruption time M to ensure that the logical error rate $e_\ell^{[M]}$ is low enough to meet the error budget. Specifically, the logical quantum computation in Eq. (3) succeeds with probability $1-\varepsilon$ provided that,

$$e_\ell^{[M]} \cdot n_{\text{patch}} \cdot t_{\text{run}} \leq \varepsilon, \tag{20}$$

i.e., the product of the logical error rate, Eq. (15), the number of patches to be decoded, Eq. (6), and the runtime of the circuit, Eq. (19), should be within the error budget ε. In the numerical experiments presented in the next section, we use a greedy approach to minimize M and d (in that order) to satisfy Eq. (20), given a input circuit and error budget.

2) Computational complexity and bandwidth: The total computational power, in FLOP/s, required for online decoding of the quantum computation can be estimated as,

$$\mathcal{C} = 10 \cdot \frac{2 \cdot n_{\text{window}}(\Delta_{\text{HSTCX}}) \cdot n_{\text{patch}} \cdot \text{FLOP}^{\text{EE}}(d)}{M \cdot t_{\text{cycle}}}. \tag{21}$$

The numerator estimates the total number of floating operations for decoding a single HSTCX-cycle using a depth-2 parallel window decoder to decode n_{patch} surface code logical qubits that each produce $n_{\text{window}}(\Delta_{\text{HSTCX}})$ syndrome windows. Here we assume that we provide sufficient decoding power to continuously process the data of all n_{patch} patches such that it is guaranteed that the size of syndrome data to be decoded when a T-gate is applied remains constant. The denominator is the time window (in seconds) during which the computation needs to be performed. The prefactor 10 is the inverse of the 10% fraction of peak performance assumed for PyMatching.

The total bandwidth required to send the syndrome data out of the dilution refrigerator to the decoder simply scales as the product of the data rate of a single logical qubit and the number of patches involved in the computation, i.e.,

$$\mathcal{B} = R_\ell \cdot n_{\text{patch}}. \tag{22}$$

V. Results

In this section, we illustrate the results obtained with the model that we developed throughout Section IV and evaluate the quantum and classical resources required for large-scale quantum computations over a broad range of parameters.

Consistent with our previous choices and aligned with common results and assumptions found in the literature, we choose a code cycle time of $t_{\text{cycle}} = 1\mu s$ and a physical error error rate of 10^{-3}. Furthermore, we assume that the decoder hardware and quantum computer are interconnected with a return latency of $\ell = 250ns$, a value representative of modern network latencies. The error budget for all considered quantum computations is set at $\varepsilon = 0.5$.

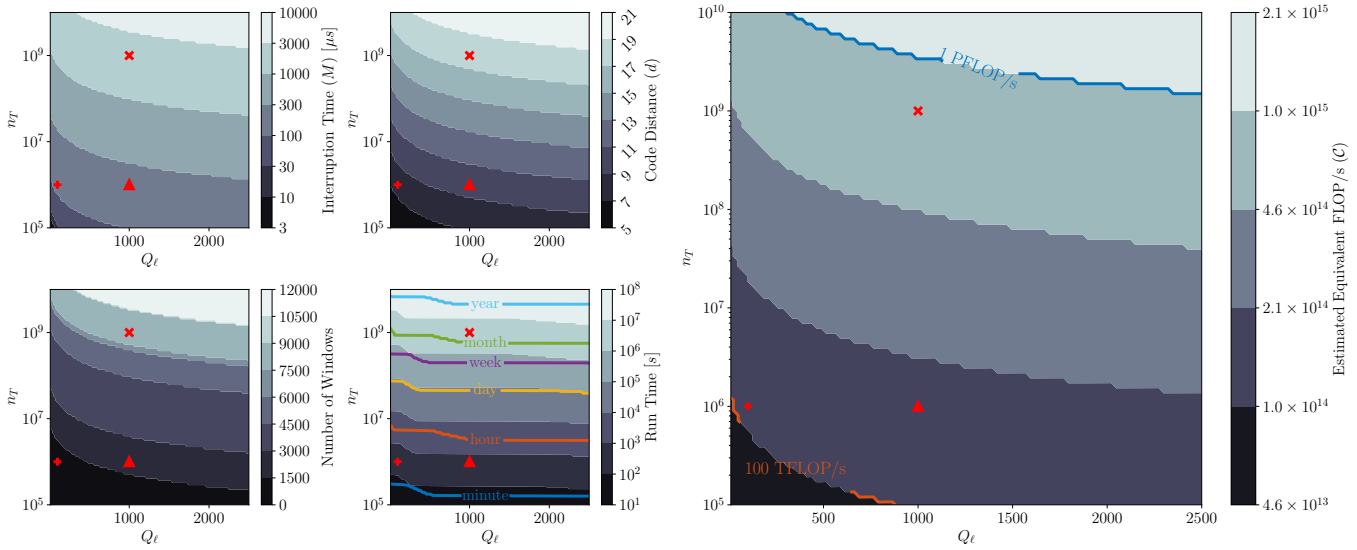

Fig. 10. Overview of the five key numerical results obtained from the model introduced in Section IV in function of the number of logical qubits $Q_\ell \in [10, 2500]$ and T-depth $n_\mathrm{T} \in [10^5, 10^{10}]$. At each coordinate (Q_ℓ, n_T), the interruption time M and the code distance d are minimized in a greedy fashion to satsify Eq. (20) with $\varepsilon = 0.5$. The $+$-, $\triangle$-, and $\times$-markers respectively indicate the coordinates $(100, 10^6)$, $(1000, 10^6)$, and $(1000, 10^9)$ which are representative for the computational complexity of applications in materials sciences [18] and biochemistry [17] that are described in the main text. **Top left:** Minimized decoder interruption time M (in μs), **Top middle:** Minimized surface code distance d, **Bottom left:** Number of decoding windows required to process the HSTCX-sequence, **Bottom middle:** Total application runtime, and **Right:** Estimated total compute intensity $\mathcal{C}$ in FLOP/s required for online decoding.

Figure 10 reports 5 key metrics, computed by the model introduced in Section IV, for quantum problems that require 10 to $2,500$ logical qubits and 10^5 to 10^{10} T-gates. At each coordinate (Q_ℓ, n_T), we minimize M and d to satisfy Eq. (20). The top left and middle panels in Fig. 10 respectively show the minimized interruption times M and code distances d. We observe that the decoding time varies between 10^{-4} and $3 \cdot 10^{-3}$ seconds for most of the parameter range, while code distances range from 5 to 21. The bottom left panel shows the number of decoding windows required to process the HSTCX-sequence data over the whole processor in a timely manner. This number is computed as $n_\mathrm{patch} \cdot n_\mathrm{window}(\Delta_\mathrm{HSTCX})$, and ranges from around 1,000 to just over 10,000 windows. The bottom middle panel shows the complete application runtime, which uses Eq. (19). The runtime starts below 1 minute and increases to over 1 year over the range of considered parameters. Finally, the right panel reports the arithmetic compute intensity $\mathcal{C}$ as estimated by Eq. (21). The required compute intensity for online decoding is less than 2.1 PFLOP/s over the full parameter range. We observe that, over the parameter range, the compute intensity has a relatively stronger dependence on the T-depth and a relatively weaker dependence on the number of logical qubits.

The red $+$-, $\triangle$-, and $\times$-markers in Fig. 10 respectively indicate the coordinates $(100, 10^6)$, $(1000, 10^6)$, and $(1000, 10^9)$. These values approximately correspond to the quantum computational resources required for computing the ground state energy of the 8×8 ($+$) and 20×20 ($\triangle$) 2D square lattice Hubbard model [18], and the chemical simulation of cytochrome P450 ($\times$), a pharmaceutically relevant molecule [17]. We observe that the Hubbard model computations each require less than 1 hour of runtime and less than 210 TFLOP/s of computational power for the decoding. The larger simulation of P450 requires more than 1 month of runtime and a sustained 1 PFLOP/s for decoding.

Figure 11 reports on the average speed per HSTCX-cycle and the syndrome data generation rates. Specifically, the left panel breaks down the duration of a full HSTCX-sequence in terms of: (i) the duration of the Clifford HSCX-sequence Δ_HSCX in blue (Eq. (4)), (ii) the duration of magic state distillation in red (Eq. (5)), (iii) the combined time of routing, CX-gate, and logical measurement for magic state injection $t_{\mathrm{inj},|T\rangle}$ in orange, (iv) the average syndrome decoding time $t_{\mathrm{dec},|T\rangle}$ in purple, and (v) the constant network latency $\ell = 250ns$ in green. We observe that the growth in average decoding time of the inner decoder according to Eq. (10) starts to dominate at relatively low code distances. At all code distances, the full time complexity of the T-gate dominates over the Clifford gates. The contribution of the network latency is negligible and cannot be visually distinguished in the figure.

The middle panel in Fig. 11 which shows the logical clock speed at which HSTCX-cycles are processed, i.e., $\Delta_\mathrm{HSTCX}^{-1}$. The logical clock speed starts at 10 kHz at distance 5, but slows down to just over 100 Hz at distance 21. This is a dramatic slowdown from the physical QPU clock speed of 1 Mhz.

The right panel in Fig. 11 shows the raw and compressed (see Eq. (2)) syndrome data generation rates for 1,000 logical qubits in function of the code distance. We observe that at distance $d = 19$, which is the predicted optimal distance for

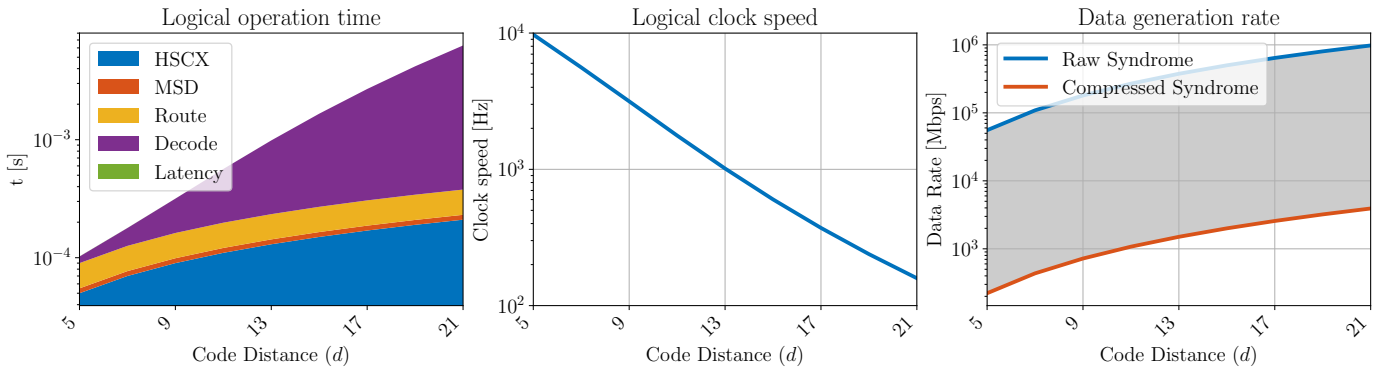

Fig. 11. **Left**: breakdown of the logical operation time of processing a single HSTCX-sequence in function of code distance d. **Middle**: Logical clock speed of the quantum computer in function of code distance d. **Right**: Raw and compressed syndrome data generation rates in Mbps in function of code distance d for $Q_\ell = 1,000$ logical qubits.

the cytochrome P450 computation [17] (see Fig. 10), either about 2 Gbps of compressed syndrome data or about 500 Gbps raw syndrome data has to be sent from the QPU to the decoder.

VI. Discussion and Conclusion

In this paper, we have developed a novel and comprehensive model that significantly extends previous work [28] and allows us to estimate the minimal required classical computational resources, including runtime, flops, and network bandwidth, to support large-scale quantum computations on a superconducting QPU equipped with a surface code. Additionally, we estimate the quantum resources required in terms of code distance and runtime, and quantify the time scales that determine the logical quantum clock speed.

The results of our study are summarized in Figs. 10 and 11 and show that at most a few PFLOP/s of classical compute power is required for online parallel window decoding over a wide range of problem sizes. This is a significant amount of computational power, but not insurmountable in comparison to the capabilities of modern HPC systems that are breaking the exascale barrier. The I/O bandwidth for online decoding is estimated to lie in the Gbit/s to Tbit/s range. By itself, this is a feasible target in light of current networking technologies. However, it has to be realized in a cryogenic environment under very limited power budgets.

As shown in the left and middle panel of Fig. 11 the overhead of quantum error correction and decoding slows down the MHz physical clock speed of superconducting QPUs to a logical clock speed of a few hundred to a few thousand Hz depending on the code distance. This is considerably slower than classical computers that can run logical operations at GHz frequencies. This gap in clock speed forms the main reason that quantum algorithmic speedups likely need to be super-quadratic to offer practical advantages [21]. Ongoing research into more efficient quantum error correction schemes [35]–[39], new qubit modalities with higher degrees of connectivity [49], and classical decoding hardware designed specifically for fast error decoding [50] are some of the most promising ways to overcome this challenge in the future.

We necessarily had to make a series of assumptions for our analysis that lead to certain limitations. We assume that parallel window decoding has perfect parallel scaling provided that sufficient resources are available. Stitching the syndrome data in round B with the committed data from round A does require communication between the processors. Further work is required to investigate how exactly this impacts the parallel scaling. Furthermore, we stick to the commonly used assumption that all T-gates are executed sequentially. This keeps the overhead for magic state distillation constant and simplifies the conceptual layout of the QPU [32]. Further work is required to investigate what impact parallel T-gates would have on our analysis. Finally, we assume a fixed logical error rate and graph-like error syndromes. This assumption is founded on theoretical error models relevant for surface code logical qubits [16]. However, experimental implementations can lead to more complicated physical error mechanisms that MWPM cannot reliably decode and instead requires computationally more intensive maximum likelihood decoders [40].

Future research avenues include extending our current framework to qLDPC codes. While the surface code can be regarded as a topological qLDPC code formed by the hypergraph product of the classical repetition code, expander qLDPC codes lead to much more efficient encoding schemes [35], [38], [39]. However, several important caveats have to be considered in such future work. First, the configuration space of expander qLDPC codes is extensive and further research is required to determine which codes meet all important key metrics such as high encoding efficiency, support for universal quantum computation, compatibility with resource state generation, and a feasible path to experimental realization given the multitude of non-local parity checks involved. Additionally the decoding algorithm is a central component of our framework and while belief propagation with ordered statistics shows promise as a decoding algorithm for qLDPC codes [51], it remains less well studied than MWPM.

Furthermore, it would be valuable to model the impact of other qubit hardware modalities, such as trapped ion and neutral atom qubits. On one hand their slower gate times

suggest that, compared to the left panel in Fig. 11, real-time decoding would become a less important component of the logical HSTCX-cycle. The estimated bandwidth and compute intensity would also decrease, while the application run time might increase. On the other hand, physical error rates of the best available trapped ion qubits are currently lower than for superconducting qubits. This implies a potential reduction in code distances and physical qubits to reach similar logical error rates. Finally, trapped ion and neutral atom systems might admit more efficient logical gate implementations, for example using transversal logical CNOT gates [49], that are enabled by their dynamic qubit connectivity. A detailed further analysis is needed to make a comparison across hardware modalities.

Data Availability

Code and data artifacts are available at https://zenodo.org/records/10818960.

References

[1] J. Shalf, "The future of computing beyond Moore's law," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, no. 2166, p. 20190061, Jan. 2020.

[2] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[3] B. Bauer, S. Bravyi, M. Motta, and G. K.-L. Chan, "Quantum algorithms for quantum chemistry and quantum materials science," *Chemical Reviews*, vol. 120, no. 22, pp. 12 685–12 717, 2020.

[4] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, M. Kieferová, I. D. Kivlichan, T. Menke, B. Peropadre, N. P. Sawaya *et al.*, "Quantum chemistry in the age of quantum computing," *Chemical reviews*, vol. 119, no. 19, pp. 10 856–10 915, 2019.

[5] A. Di Meglio *et al.*, "Quantum Computing for High-Energy Physics: State of the Art and Challenges. Summary of the QC4HEP Working Group," 7 2023.

[6] T. S. Humble *et al.*, "Snowmass White Paper: Quantum Computing Systems and Software for High-energy Physics Research," in *Snowmass 2021*, 3 2022.

[7] O. Philipsen, "Constraining the phase diagram of qcd at finite temperature and density," in *Proceedings of 37th International Symposium on Lattice Field Theory — PoS(LATTICE2019)*, ser. LATTICE2019. Sissa Medialab, Jan. 2020.

[8] E. Poppitz and Y. Shang, "Chiral lattice gauge theories via mirror–fermion decoupling: A mission (im)possible?" *International Journal of Modern Physics A*, vol. 25, no. 14, pp. 2761–2813, 2010.

[9] M. L. Wagman and M. J. Savage, "Statistics of baryon correlation functions in lattice qcd," *Phys. Rev. D*, vol. 96, p. 114508, Dec 2017.

[10] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver, "Superconducting qubits: Current state of play," *Annual Review of Condensed Matter Physics*, vol. 11, no. 1, pp. 369–395, 2020.

[11] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-ion quantum computing: Progress and challenges," *Applied Physics Reviews*, vol. 6, no. 2, 2019.

[12] T. M. Graham, Y. Song, J. Scott, C. Poole, L. Phuttitarn, K. Jooya, P. Eichler, X. Jiang, A. Marra, B. Grinkemeyer, M. Kwon, M. Ebert, J. Cherek, M. T. Lichtman, M. Gillette, J. Gilbert, D. Bowman, T. Ballance, C. Campbell, E. D. Dahl, O. Crawford, N. S. Blunt, B. Rogers, T. Noel, and M. Saffman, "Multi-qubit entanglement and algorithms on a neutral-atom quantum computer," *Nature*, vol. 604, no. 7906, p. 457–462, Apr. 2022.

[13] B. M. Terhal, "Quantum error correction for quantum memories," *Rev. Mod. Phys.*, vol. 87, pp. 307–346, Apr 2015.

[14] P. Shor, "Fault-tolerant quantum computation," in *Proceedings of 37th Conference on Foundations of Computer Science*, 1996, pp. 56–65.

[15] S. B. Bravyi and A. Y. Kitaev, "Quantum codes on a lattice with boundary," 1998.

[16] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, 08 2002.

[17] 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," *Proceedings of the National Academy of Sciences*, vol. 119, no. 38, p. e2203533119, 2022.

[18] E. T. Campbell, "Early fault-tolerant simulations of the hubbard model," *Quantum Science and Technology*, vol. 7, no. 1, p. 015007, Nov. 2021.

[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," 2022.

[20] S. S. Tannu, Z. A. Myers, P. J. Nair, D. M. Carmean, and M. K. Qureshi, "Taming the instruction bandwidth of quantum computers via hardware-managed error correction," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 679–691.

[21] T. Hoefler, T. Häner, and M. Troyer, "Disentangling hype from practicality: On realistically achieving quantum advantage," *Communications of the ACM*, vol. 66, no. 5, p. 82–87, Apr. 2023.

[22] A. Holmes, Y. Ding, A. Javadi-Abhari, D. Franklin, M. Martonosi, and F. T. Chong, "Resource optimized quantum architectures for surface code implementations of magic-state distillation," *Microprocessors and Microsystems*, vol. 67, pp. 56–70, 2019.

[23] O. Higgott and C. Gidney, "Sparse blossom: correcting a million errors per core second with minimum-weight matching," 2023.

[24] Y. Wu and L. Zhong, "Fusion blossom: Fast MWPM decoders for qec," 2023.

[25] X. Tan, F. Zhang, R. Chao, Y. Shi, and J. Chen, "Scalable surface code decoders with parallelization in time," 2022.

[26] L. Skoric, D. E. Browne, K. M. Barnes, N. I. Gillespie, and E. T. Campbell, "Parallel window decoding enables scalable fault tolerant quantum computation," *Nature Communications*, vol. 14, no. 1, Nov. 2023.

[27] H. Bombín, C. Dawson, Y.-H. Liu, N. Nickerson, F. Pastawski, and S. Roberts, "Modular decoding: parallelizable real-time decoding for quantum computers," 2023.

[28] N. Delfosse, A. Paz, A. Vaschillo, and K. M. Svore, "How to choose a decoder for a fault-tolerant quantum computer? The speed vs accuracy trade-off," 2023.

[29] P. Das, C. A. Pattison, S. Manne, D. M. Carmean, K. M. Svore, M. Qureshi, and N. Delfosse, "Afs: Accurate, fast, and scalable error-decoding for fault-tolerant quantum computers," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 259–273.

[30] N. Delfosse, "Hierarchical decoding to reduce hardware requirements for quantum computing," 2020.

[31] S. C. Smith, B. J. Brown, and S. D. Bartlett, "Local predecoder to reduce the bandwidth and latency of quantum error correction," *Phys. Rev. Appl.*, vol. 19, p. 034050, Mar 2023.

[32] D. Litinski, "A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery," *Quantum*, vol. 3, p. 128, Mar. 2019.

[33] H. Bombin and M. A. Martin-Delgado, "Topological quantum distillation," *Physical review letters*, vol. 97, no. 18, p. 180501, 2006.

[34] A. Y. Kitaev, "Fault-tolerant quantum computation by anyons," *Annals of physics*, vol. 303, no. 1, pp. 2–30, 2003.

[35] P. Panteleev and G. Kalachev, "Asymptotically good quantum and locally testable classical ldpc codes," in *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 375–388.

[36] A. Strikis and L. Berent, "Quantum low-density parity-check codes for modular architectures," *PRX Quantum*, vol. 4, p. 020321, May 2023.

[37] N. Breuckmann and J. Eberhardt, "Balanced product quantum codes," *IEEE Transactions on Information Theory*, vol. 67, no. 10, pp. 6653 – 6674, Jul. 2021.

[38] J.-P. Tillich and G. Zémor, "Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength," *IEEE Transactions on Information Theory*, vol. 60, no. 2, pp. 1193–1202, 2014.

[39] A. Leverrier, J.-P. Tillich, and G. Zémor, "Quantum expander codes," in *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, 2015, pp. 810–824.

[40] G. Q. A. Team, "Suppressing quantum errors by scaling a surface code logical qubit," *Nature*, vol. 614, no. 7949, pp. 676–681, Feb. 2023.

[41] D. Horsman, A. G. Fowler, S. Devitt, and R. V. Meter, "Surface code quantum computing by lattice surgery," *New Journal of Physics*, vol. 14, no. 12, p. 123011, Dec. 2012.
[42] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of Mathematics*, vol. 17, p. 449–467, 1965.
[43] A. G. Fowler and C. Gidney, "Low overhead quantum computation using lattice surgery," *arXiv preprint arXiv:1808.06709*, 2018.
[44] C. Gidney, "Stim: a fast stabilizer circuit simulator," *Quantum*, vol. 5, p. 497, Jul. 2021.
[45] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits," *Phys. Rev. A*, vol. 70, p. 052328, Nov 2004.
[46] D. Litinski and F. v. Oppen, "Lattice Surgery with a Twist: Simplifying Clifford Gates of Surface Codes," *Quantum*, vol. 2, p. 62, May 2018.
[47] S. Bravyi and A. Kitaev, "Universal quantum computation with ideal Clifford gates and noisy ancillas," *Phys. Rev. A*, vol. 71, p. 022316, Feb 2005.
[48] O. Higgott, "Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, jun 2022.
[49] D. Bluvstein, S. J. Evered, A. A. Geim, S. H. Li, H. Zhou, T. Manovitz, S. Ebadi, M. Cain, M. Kalinowski, D. Hangleiter, J. P. B. Ataides, N. Maskara, I. Cong, X. Gao, P. S. Rodriguez, T. Karolyshyn, G. Semeghini, M. J. Gullans, M. Greiner, V. Vuletić, and M. D. Lukin, "Logical quantum processor based on reconfigurable atom arrays," *Nature*, Dec. 2023.
[50] B. Barber, K. M. Barnes, T. Bialas, O. Buğdaycı, E. T. Campbell, N. I. Gillespie, K. Johar, R. Rajan, A. W. Richardson, L. Skoric, C. Topal, M. L. Turner, and A. B. Ziad, "A real-time, scalable, fast and highly resource efficient decoder for a quantum computer," 2023.
[51] J. Roffe, D. R. White, S. Burton, and E. Campbell, "Decoding across the quantum low-density parity-check code landscape," *Phys. Rev. Res.*, vol. 2, p. 043423, Dec 2020.