Control Requirements and Benchmarks for Quantum Error Correction

Yaniv Kurman[1], Lior Ella[1], Ramon Szmuk[1], Oded Wertheim[1], Benedikt Dorschner[2], Sam Stanwyck[2], and Yonatan Cohen[1]

[1]*Quantum Machines Inc., Tel Aviv, Israel*

[2]*NVIDIA Corp, Santa Clara, CA, USA*

Reaching useful fault-tolerant quantum computation relies on successfully implementing quantum error correction (QEC). In QEC, quantum gates and measurements are performed to stabilize the computational qubits, and classical processing is used to convert the measurements into estimated logical Pauli frame updates or logical measurement results. While QEC research has concentrated on developing and evaluating QEC codes and decoding algorithms, specification and clarification of the requirements for the classical control system running QEC codes are lacking. Here, we elucidate the roles of the QEC control system, the necessity to implement low latency feed-forward quantum operations, and suggest near-term benchmarks that confront the classical bottlenecks for QEC quantum computation. These benchmarks are based on the latency between a measurement and the operation that depends on it and incorporate the different control aspects such as quantum-classical parallelization capabilities and decoding throughput. Using a dynamical system analysis, we show how the QEC control system latency performance determines the operation regime of a QEC circuit: latency divergence, where quantum calculations are unfeasible, classical-controller limited runtime, or quantum-operation limited runtime where the classical operations do not delay the quantum circuit. This analysis and the proposed benchmarks aim to allow the evaluation and development of QEC control systems toward their realization as a main component in fault-tolerant quantum computation.

Introduction

Quantum error correction (QEC) [1,2] stands as the clearest path for reaching the exponential advantage of quantum computing and solving problems of great significance such as simulating complex quantum systems [3], factorization [4], and more. In expected QEC implementations, the quantum information is encoded over several physically separated qubits, and local parity measurements (stabilizers) which enable error detection and correction of local physical errors [5]. Although adding physical qubits adds errors, increasing the number of qubits will reduce exponentially the logical quantum errors if the physical errors are below a certain QEC code-dependent threshold [6,7], opening the path toward useful quantum computation.

Apart from the requirements on the quantum hardware, the successful execution of QEC codes depends on the classical control system. The classical control system is responsible for all classical aspects of quantum computing and includes executing the quantum control and measurement signals, acquiring the readout signals, and performing classical processing operations [8], all in a synchronized manner. In QEC, the control system is also responsible for mapping the local physical measurements into logical measurement results or logical Pauli frame updates [5,9–11] through an algorithmic procedure called decoding [12]. To reach useful quantum computation with QEC, i.e., perform fault-tolerant non-Clifford gates, the control unit is required to perform quantum gates that depend on the decoding output. Moreover, to prevent a diverging classical calculation latency, it is crucial for the control system to close a tight loop, with ultra-low latency, from the physical quantum measurement through the classical decoding procedure to a conditional feed-forward quantum operation [13]. Thus, the success of fault-tolerant quantum computation depends on minimizing the QEC *feed-forward latency*.

Here, we analyse the requirements of a control system executing QEC codes, clarify the need for low latency feed-forward capabilities, and propose control system benchmarks for its ability to run fault-tolerant quantum computation. The proposed benchmarks are based on two feed-forward operations representing the necessity to run stabilizer rounds in parallel to the decoding and feed-forward operation. As a result, additional data is generated during the decoding, creating a heavier computational load on the consequent decoding and feed-forward operations. Instead of the common approach of evaluating solely the decoder performance, we define the benchmarks as the time between two analogue signals to include all classical components, from RF quantum control to controller-decoder communication, the decoding,

and parallelization of the above. As quantum computers mature, the ability to evaluate holistically the control hardware with simple benchmarks is instrumental for the development of QECCPs (Quantum error correction control processor) toward the realization of quantum advantage.

Recent years have witnessed significant scientific and engineering advancements in the realm of Quantum Control Processor (QCP) designs and quantum-classical processing capabilities [8,14,15], enabling the realization QECCPs. State-of-the-art QCPs support real-time control operations such as conditional pulses [16–19], real-time control flow [20], parameter updates using reinforcement learning [21,22] or model-based optimizations [23], as well as comprehensive embedded calibration and workflows [24–27]. These capabilities have been instrumental in optimizing quantum fidelities, which, among other groundbreaking quantum research, enabled variational quantum computation [28], extending the qubit lifetimes [16,21], flagged-syndrome-based fault-tolerant quantum circuits [20], and reaching the error-threshold point for QEC surface codes [11]. We explain below that for scalable QEC computation for practical applications, the QECCP must encompass all the mentioned QCP capabilities, in addition to enhanced classical calculations, including the execution of error decoding algorithms within s timeframes.

Our manuscript is structured as follows. In Section II we give an example of the required tasks of the QECCP based on the well-known surface code. Section III is devoted to explaining in detail why the QEC feed-forward latency will determine the logical clock cycle and whether executing QEC quantum calculations is possible. In section IV, we suggest two benchmarks, which are defined by pseudocode and exemplified by simple logical operations. Finally, in section V we add a discussion about additional possible benchmarks and the long-term requirements of a QECCP once quantum hardware is scaled.

II. The QECCP required operations, exemplified with a surface code

Among the different possible QEC codes that can be used to benchmark and evaluate the QECCP, the surface code [9,29–31] stands out because of its high error threshold of ~1%, orders of magnitude higher than other codes, and clear physical implementation (requires nearest-neighbour connectivity). Due to these qualities, many significant aspects of the surface code have been developed in detail, including fault-tolerant computation techniques [32–35], high-fidelity magic state preparation [36–38], decoding algorithms [39–46], simulation

tools [47], distillation schemes [30,48–50], and scaling suggestions [30,35]. The clear path to implement fault-tolerant quantum computing with surface codes motivates us to define the QECCP requirements with the surface codes as a representative example, though the QECCP quantum and classical operations, as well as the benchmarks below, are relevant to any stabilizer code.

In brief, the surface code implements each logical qubit with a set of parity checks that can be tiled onto a square lattice. In the qubit-efficient rotated surface implementation, the quantum information of a single qubit (logical qubit) is encoded in a single surface with d^2 physical qubits called data qubits (black circles in Fig. 1a), where d is the distance of the QEC code. In addition to the data qubits, the logical qubit requires $d^2 - 1$ ancillary physical qubits (green and red circles in Fig. 1a), which perform local parity checks (called a stabilizer) of their nearest-neighbour data qubits. The stabilizers are the heart of any stabilizer-based QEC code: the set of stabilizer operators defines the Hilbert subspace of the logical qubit, called the code space [5]. The stabilizer measurements collapse the quantum state into the code space up to a local Pauli error which is detected according to the measurement outcome (called a syndrome). The desired quantum information is encoded by initialization of the data qubits followed by a measurement of all stabilizers (stabilizer round, Fig. 1b), while the quantum information is stored when repeating the stabilizer measurements and keeping track of the syndrome measurement results, in a process called decoding.

Figure 1: **Example of a distance-3 surface code performing a non-Clifford gate. (a)** A logical qubit in the surface code, implemented with d^2 data qubits (black) and $d^2 - 1$ ancilla qubits (red and green) which are used for the stabilizer measurements. **(b)** The physical quantum circuit in an 8-step stabilizer measurement round for the X (top) and Z (bottom)

stabilizers. The controller is required to perform parallel multi-qubit operations in each step (some operations are shown in (a)), for example, measure all ancillary qubits of the surface in parallel in the final step. **(c)** The physical implementation of a non-Clifford gate with surface codes. Initialization of the data qubits of an ancillary surface to implement a magic state (top), a lattice surgery between the ancillary surface and a computational qubit (middle) where the two surfaces are converted to one elongated surface (which ends by measuring the surgery data qubits), and a measurement of the data qubits of the ancillary surface (bottom). We note that these operations are separated by stabilizer rounds. **(d)** The logical circuit that describes the operations in (c), performing a non-Clifford gate. The executed gate is determined by the logical measurement results. Feed-forward in the form of a lattice surgery with a Pauli state is required for executing the planned logical gate. **(e)** The quantum logic that the QECCP is required to execute.

The fault-tolerant non-Clifford gates with surface codes are done in a measurement-based quantum computation fashion. Such quantum computation can be done by preparing an ancillary qubit in a magic state, entangle it with the computational qubits, measure the ancillary qubit, and apply *feed-forward* according to the measurement result [51]. The feed-forward is required to apply a specific non-Clifford gate of choice. To execute this procedure with surface codes, the quantum logic that the classical controller should perform includes (Fig. 1e and examples in Fig. 1c-d):

1. Logical qubit initialization: encoding within the data qubits a specific logical state (for example in the magic state $|T\rangle_L = |0\rangle_L + e^{i\pi/4}|1\rangle_L$ [36–38,52]).

2. Lattice surgery: combining two or more surfaces into one elongated surface. The measurement of ancillary qubits that are located between two (or more) surfaces [32,53] performs a logical multi-qubit Pauli measurement (for example, M_{XX} or M_{ZZ}) which reduces the Hilbert space by a degree of freedom.

3. Single logical qubit measurement: measurement of the surface's data qubits in the X or Z basis.

4. Feed-forward logical operations: based on the QEC decoding, a real-time logical circuit modification (which include the logic operations in 1-3) is needed to implement the desired non-Clifford logical operation.

In the example shown in Fig. 1d, to perform a $T = diag(1, e^{i\pi/4})$ gate at the logical level, the feed-forward logical gate get the form of a logical $S = diag(1, i)$ gate which is performed by a lattice surgery with a surface in a Pauli state that is conditioned to the first lattice surgery result. Since this correction cannot propagate through a non-Clifford gate (as we explain in SM section S1), a useful quantum circuit which is formed by a set of non-Clifford gates (each is executed through a procedure similar to Fig. 1d) will be delayed until the feed-forward is applied.

The feed-forward gate will depend on the QEC decoding outcome. The decoding unit (decoder) processes all measurements that were acquired up to a certain point and converts this data into a logical Pauli frame flip ($I, X, Y,$ or Z) of each logical qubit and determines logical measurement results (single-qubit measurements and lattice surgeries). The logical Pauli frame corrections will not necessarily lead to any additional physical gates (that can introduce additional errors and delay the circuit) since the Pauli corrections can propagate in software to the consequent logical measurement [29]. However, since a feed-forward operation is inherent to the non-Clifford gate implementation, a full QECCP is required to support a hardware-efficient and accurate decoder with low-latency for the logical feed-forward.

The final part of the QECCP will be the orchestration between all operations above. We present in Fig. 2 a possible architecture and data flow of the QECCP. Once a quantum logical gate-level program is defined, and before its execution (offline), the QECCP translates the logical level circuit to surface operations [54] and then to a pulse-level program that includes an optimized pulse sequence, classical decision-making, and conditional gates. This program is then loaded into the quantum processing unit (QPU) controller. Additionally, the decoder receives the expected data transfer, communication protocol with the QPU controller, the decoding algorithm, and the decoding parameters, e.g., the matching graph dimensions that can be static or dynamic. Then, the circuit execution starts (runtime). The physical quantum measurement results (ancillary and data-qubit measurements) are passed to a decoder that returns logical frames and logical measurement results to the controller and the user. The benchmarks below attempt to verify the capability of a QECCP to execute all of the above while evaluating the feed-forward latency.

Figure 2. QECCP architecture and dataflow. Offline, the logical gate-level circuit is defined, from which the physical pulse sequence and decoding operations are derived (e.g., a matching graph). During runtime, the physical circuit is executed with syndrome flow from the

controller to the decoder while the decoding results in the form of logical measurement results and logical frames are returned to the QPU controller and sent to the user.

III. Feed-forward latency requirements

One of the greatest challenges in performing QEC is reaching low feed-forward latencies, that is, a short time from the physical execution of a logical measurement until the controller plays a conditional pulse which depends on the logical measurement outcome. The necessity for feed-forward arises from the requirement to perform non-Clifford gates to reach quantum advantage (Gottesman-Knill theorem) [55]. Since some Pauli errors cannot propagate through non-Clifford gates without being converted into non-Clifford errors (see SI section S1), it is mandatory to correct the logical frame flips of the QEC code or modify the logical circuit before the error propagates. Therefore, the conditional *feed-forward of each non-Clifford gate* must depend on the decoding result (explained in section II), where a correction must be applied before the consequent non-Clifford gate is fully executed. In this chapter, we provide a general analysis of the feed-forward latency under different classical parameters. We note that the analysis in this chapter can be implemented for any QEC stabilizer code that requires feed-forward to execute fault-tolerant quantum computation.

To exemplify the effect of a delayed feed-forward, we consider a fault-tolerant QEC circuit of two consecutive non-commuting non-Clifford gates (Fig. 3a) with limited quantum resources of a single ancillary surface. In the logical circuit level (Fig. 3b), the execution of the circuit uses the ancillary logical qubit (surface) to implement the non-Clifford gates and their conditional corrections in a fault-tolerant manner. The physical implementation of the circuit is illustrated in a two-dimensional (space-time) view in Fig. 3c. Since the correction of each non-Clifford gate depends on a different decoding result, the physical measurements in space and time should be divided to different decoding tasks (each decoding tasks is represented by a different colour in Fig. 3c). Although most available decoders do not support the ability to divide the syndromes into different decoding tasks (with joint boundaries), without it, it is not possible to implement sequential non-Clifford gates. Specifically, the output of the decoding tasks that include lattice surgery between the computational surface and a magic state will cause a conditional modification to the quantum circuit. In this example, the outcome of the first decoding task (yellow in Fig. 3c) will determine the initialization state of the ancillary surface, either $|i\rangle = |0\rangle + i|1\rangle$ to perform the S gate correction or $|X^{\frac{1}{4}}\rangle = |+\rangle + e^{i\pi/4}|-\rangle$ to start the next non-Clifford gate.

Figure 3. Example of non-Clifford computation with surface codes. (a) An example of a logical circuit containing two non-Clifford gates. **(b)** The fault-tolerant logical circuit that implements the circuit in (a) with surface codes with only a single ancillary surface. The dashed square denotes the feed-forward conditional logical gates that verify that the planned circuit is executed. **(c)** The space-time view of the circuit in (b) with surface codes. Each colour denotes a separate decoding task, chosen to end each task with a logical measurement. The decoding outcome of the lattice surgery between a magic state surface and the computation surface determines a feed-forward circuit, which delays the circuit by if the feed-forward latency (L) is larger than a threshold latency L_{th}. We note that the boundary conditions between decoding tasks are necessary to match (red lines) syndromes (red Xs) between tasks. An implementation of a similar logical circuit with additional ancillary surfaces using the autocorrected $\pi/8$ technique [30], which can reduce the logical clock time while increasing L_{th}, is shown in Figs. S1 and S2.

Fig. 3c also presents examples where extended feed-forward latencies ($L^{(0)}$ and $L^{(1)}$) delay the logical clock ($\delta^{(0)}$ and $\delta^{(1)}$ respectively). However, not all values of feed-forward latencies will necessarily delay the logical gate execution since several stabilizer rounds prior to a logical measurement are needed to overcome the measurement errors. That is, the feed-forward to the system cannot be applied immediately for correcting the quantum state (as in [16,20,21,56]). The number of stabilizer rounds before the logical measurement (r_m) defines a threshold latency $L_{th} = T_s r_m$ (with T_s being the stabilizer cycle time). If the controller is ready to apply the feed-forward before L_{th}, then the classical controller will wait for the ancillary surface to be measured before it can be initialized again. This regime is quantum-operation limited where the classical hardware does not delay the circuit and the logical clock cycle is completely determined by quantum operations. Once the feed-forward latency (in some decoding task n) is larger than the threshold latency $L^{(n)} > L_{th}$, the classical control delays the circuit, causing one of the following options: (i) the steady-state feed-forward latency (L_{ss}) will

return to L_{th} leading to a *quantum-operation limited* regime, (ii) the system will reach a steady-state feed-forward latency larger than the threshold latency $L_{ss} > L_{th}$ (*classical-operation limited* regime), or (iii) the feed-forward latency will continuously increase to a *latency divergence* regime $\lim_{n\to\infty} L^{(n)} \to \infty$ which is infeasible for useful quantum computation.

In order to elucidate the origins of the various behavioural patterns, we conduct a dynamical system analysis and introduce communication theory terminology. $L^{(n)}$ is connected to the number of syndromes that the decoder is required to analyze in task n, $N^{(n)}$, and the decoding latency, $L_{dec}(N)$, through $L^{(n)} = \max(L_{th}, L_{dec}(N^{(n)}))$. The system becomes dynamical since $N^{(n+1)}$ depends on $L^{(n)}$ through $N^{(n+1)}(L^{(n)}) = N_0 + \lambda(L^{(n)} - L_{th})$, where N_0 denotes the average number of syndromes when $L^{(n)} = L_{th}$ and λ is the syndrome arrival rate to the decoder (in units of syndromes per second). As a result, the system has a non-linear connection between $L^{(n+1)}$ and $L^{(n)}$. Interestingly, increasing L_{th} without increasing the logical clock can be done with sufficient quantum resources, for example with the auto-corrected $\pi/8$ scheme [30] (see SI section 2 and Figs. S1,S2); however, increasing L_{th} artificially by additional stabilizer rounds will cause extended logical clock cycles, causing a larger N_0 and a smaller logical-coherence times.

In Fig. 4a, we show examples of system dynamics for a linear decoder and a varying complexity decoder, when plotting on the same plot $N(L)$ and $L_{dec}(N)$. The system's initial state, $L_{dec}(N_0)$, evolves (arrows) until reaching a steady-state point where the two curves intersect. In systems where the two plots do not intersect, the system will not reach a steady-state and the latency will diverge so that no useful computation could be performed. This effect can be understood through a linear decoder model $L_{dec}(N) = \tau_0 + N/T$ where T is the throughput of the decoder (in units of decoded syndromes per second) and τ_0 is a latency offset term that includes the decoder-controller communication and decoder bring-up latencies. If $L_{dec}(N_0) > L_{th}$, an intersection between the linear decoding latency $L_{dec}(N)$ and $N(L)$ can be reached only if the ratio $U = \lambda/T$ is smaller than 1. This ratio, of the arrival rate over the throughput has a significant meaning in communication theory, describing the *utilization of the decoder*. For a general decoding latency, the system will converge into a steady-state if $U(N) = \lambda/T(N) < 1$ for all N where the throughput will be dependent on N through $T(N) = \frac{\partial N}{\partial L_{dec}} = \left(\frac{\partial L_{dec}(N)}{\partial N}\right)^{-1}$. Additionally, we note that the intersections between the curves can be a stable steady-state point or an unstable point, depending if $U < 1$ or $U > 1$ at the intersection point,

respectfully. By utilizing these plots, one can assess the system dynamics in the presence of significant fluctuations in latency or syndromes, employing a probabilistic methodology to examine the likelihood of latency divergence. In essence, these graphical representations allow the determination of the operational boundaries for QECCP with any quantum hardware and any QEC codes.

Figure 4. Simulations of the feed-forward latency, and computation regimes. (a) A dynamical system analysis for a general decoder, presenting the dynamics of the system using the curves of $L_{dec}(N)$ (blue and red) to $N(L)$ (purple) from the initial state of $L_{dec}(N_0)$ until reaching a steady-state when the curves intersect. The latency can also diverge when the intersection does not exist or beyond an unstable steady-state point. **(b-c)** The steady-state feed-forward latency for different decoding parameters (b), for a linear decoding model ($\alpha = 1$, c). **(d)** The feed-forward latency ratio (FLR, d). The parameter choice can cause different operation regimes: Quantum operation limited (if $L_{ss} = L_{th}$), classical operation limited (if $L_{th} < L_{ss} < \infty$), or latency divergence where the classical operation will prevent any useful quantum computations ($FLR(n \to \infty) > 1$). All simulation parameters are presented in Table S1, and latency plots for various values of L_{th} are presented in Fig. S3.

In realistic decoders, numerical analysis shows that the decoding latency behaves as $L_{dec}(N) = \tau_0 + \tau_1 N^\alpha$ (for example, [44,45]), where α is the complexity factor of the decoder, and τ_1 is a pre-factor so that $\lambda \tau_1$ describes the linear decoder utilization (U_{lin}). Fig. 4b presents the steady-state feed-forward latency, which we calculate recursively as $L^{(n>0)} = \max(\tau_0 +$

$\tau_1 \left(N_0 + \frac{U_{lin}}{\tau_1} \left(L^{(n-1)} - L_{th} \right) \right)^\alpha$, L_{th}), showing numerically how the decoder's complexity and linear utilization determine the different regimes of operation.

The different regimes of the QEC quantum calculation can be derived analytically under a linear decoder assumption (plotted in Fig. 4c). When the utilization is $U > 1$ (red-zone), the latency *diverges* so that the classical behaviour prevents any quantum calculation. When $U < 1$, the steady-state feed-forward latency is calculated (in SI section S3) through eq. (S2) as

$$L_{ss} = L_{th} + (L(N_0) - L_{th}) \frac{1}{1-U} \Theta(L(N_0) - L_{th}) \tag{1}$$

with $\Theta(\ )$ being the step function. In this case, the steady-state circuit delay per logical gate due to the classical hardware will be $(L(N_0) - L_{th}) \frac{1}{1-U} \Theta(L(N_0) - L_{th})$. If $L(N_0) > L_{th}$, the steady-state latency will grow with the utilization and diverge as the utilization approaches 1. Although this case is within the classical-control limited regime, which can potentially support quantum computation, the logical lifetimes of the logical qubits will limit the capability to execute the logical quantum circuit.

For an unknown QECCP performance, a simple but general method to determine experimentally if the system will reach a divergence regime is by examining the feed-forward latency ratio (FLR). This factor is defined as $FLR(n)=L^{(n)}/L^{(n-1)}$, denoting the ratio between the latencies of successive logical gates, shown in Fig. 4d. If the FLR converges to a value larger than one then the latency diverges, but if the FLR converges to 1 the QECCP can potentially support quantum calculations in either the quantum or classical control limited regime, even if with a large logical clock cycle.

IV. QECCP benchmarks: feed-forward latency and latency ratio

After recognizing the pivotal influence of classical control and computational performance on fault-tolerant QEC computations, we propose two benchmarks aimed at evaluating the QECCP capability to support such computations in the near and medium term. These benchmarks focus on *Latency benchmarking*. Specifically, the duration it takes from a set of measurements that implement a *logical measurement* until a conditional set of feed-forward operations is performed to implement a *logical feed-forward* on a *logical qubit*, based

on the error decoding of a *logical observable*. Each latency benchmark below relates to two decoding tasks that encompass in a single parameter three key aspects: (i) the feed-forward latency from the last input of measurement signals to the controller until the first conditional output signal from the controller; (ii) the capacity to simultaneously execute quantum operations alongside the decoding process; and (iii) the decoding time of syndromes which were created during previous decoding tasks (i.e., the decoding throughput). The experiments that define the benchmarks below can be extended beyond two decoding tasks to check the operation regime of the classical hardware.

To create well-defined and rigorous benchmarks, we focus on a specific *representative* configuration: a QEC rotated surface code of distance-5 and distance-11 with a stabilizer round cycle time of 1μs, a union-find decoding algorithm [40], physical error rates of 0.5% (two-qubit depolarization and single-qubit measurement), and a logical circuit that is uniquely defined in each of the two benchmarks below. We require that each physical measurement will be initiated by randomly generated RF pulses according to the error model so that the benchmark will include the signal processing involved in typical superconducting transmon state estimation (demodulation, integration, and threshold-based discrimination). The benchmarks can then be evaluated using the control system only, without a connection to quantum hardware. This representative use case pushes the classical hardware to the limit since superconducting qubits have the fastest clock cycle today, orders of magnitude faster than the readout time of other implementations (such as atoms, ions, and semiconductor defect spins). We believe that these benchmarks are relevant even for the slower implementations, as we expect gate times to drop as the field progresses. Finally, we define a specific decoding algorithm, which we chose as a representative of the widely-used minimum-weight-perfect-matching variants. It is important to predefine a decoding algorithm for the benchmarks so that different controllers would perform a similar classical calculation, i.e., will generate the same output given a similar input.

Near-term benchmark: Two Intershot Feed-forward Latency (TIFL)

The first benchmark is based on a toy use case in which the quantum resources are limited to a single logical qubit to check the QECCP capabilities for running real-time single-surface experiments that require real-time decoding. The logical circuit implemented in the benchmark is shown in Fig. 5a. The controller initializes the surface (data qubits) in the $|0\rangle$ state, performs 10 stabilizer rounds, and then measures the data qubits. The benchmark clock

starts once the last sample of the last data qubit measurement signal is sampled by the controller (as the measurement timestamp in [8]). The controller then performs the surface initialization again for the next shot followed by execution of stabilizer rounds in a repeat-until-success fashion until the decoding of the first shot is done. Then, the controller applies the appropriate conditional feedback based on the decoded frame and measures the surface again. We measure latency of the first decoding task, denoted by L^0, as the time from the start of the benchmark clock until the first sample of the conditional pulse is played from the controller (as the conditional pulse timestamp in [8]).

Figure 5. The Two Intershot Feed-forward Latency (TIFL) benchmark. (a) Logical circuit for measuring the TIFL benchmark. The double line represents a conditional gate, and the red double arrow represents the experimental feed-forward latency of two shots, and the TIFL benchmark. The stabilizer rounds are not shown as they represent a logical identity. **(b)** Surface view of the TIFL benchmark, where the data for the first decoding task (yellow) is predetermined in the benchmark as 10 stabilizer rounds, while the number of stabilizer rounds in the second decoding task (orange) is determined by the first decoding latency (L^0).

To further verify the scalability of the classical hardware, we do not stop the benchmark once the first feed-forward latency is measured (related to the decoding of 10 stabilizer rounds). We set the end of the benchmark to be the moment the *second* feed-forward analogue pulse is applied. This second feed-forward has a latency of L^1, related to the decoding task which contains an unknown number of stabilizer rounds that were measured during the first feed-forward. Since the benchmark relates to the time between two decoding tasks within a single surface 49 (distance-5) or surface 241 (distance-11), we call these benchmarks Two Intershot Feed-forward Latency 49 or 241, or in short, the TIFL-49 or TIFL-241 benchmarks.

The concrete definition of the TIFL benchmark is shown in Listing 1 as a pseudocode. We write the variables and building block commands required to run a single-surface memory experiment and measure the benchmark. The pulse-level statements and macros "**initialize surface**", "**play**", "**stabilizer round**", and "**measure surface**" include a set of predefined physical RF pulses and measurements that are detailed in Listing S1 and are compatible with the representative use case discussed above. To create the separation between the classical hardware that we want to benchmark and the quantum hardware (that is, to be able to benchmark the classical hardware without quantum hardware), we add as an input to these operations the predefined parameters **error_probability** and **round_time** for the stabilizer round that we can control classically in the analogue input to the controller. Interestingly, although we aim for a surface code, the pseudocode in Listing 1 can be used to benchmark any QEC stabilizer code.

```
rounds = 10
error_probability = 0.01
round_time = 1E-6
decoding_algorithm = Union_find
send_to_decoder(algorithm=decoding_algorithm)

initialize_surface(q0, state=1)
for i in range(rounds):
    ancilla_bits = stabilizer_round(q0, error_probability, round_time)
    send_to_decoder(ancilla_bits, task=0)

data_bits = measure_surface(q0, error_probability, timestamp->tic)
for i in [0,1]:
    send_to_decoder(data_bits, task=i)
    initialize_surface(q0, state=i)
    [logical_result, decoding_recieved] = get_decoding_result(task=i)
    while not decoding_recieved:
        ancilla_bits = stabilizer_round(error_probability, round_time)
        send_to_decoder(ancilla_bits, task=i+1)
        [logical_result, decoding_received] = get_decoding_result(task=i)
    if logical_result:
        play_x(q0)
    if i == 0:
        data_bits = measure_surface(q0, error_probability)
    else:
        toc = timestamp

TIFL= toc - tic
```

Listing 1: Pseudocode for measuring the intershot feed-forward latency. Green: elements of the configuration file. **Blue**: real-time classical variables. **Red**: pulse-level commands. **Orange**: pre-defined constants. **Bright Purple**: control flow statement. **Cyan**: controller-decoder communication statement. **Mustard**: decoding algorithm. We note that the macros "**stabilizer_round**", "**initialize_surface**", and "**measure_surface**" are defined in Listing S1.

Apart from the analogue signals, we write the controller-decoder communication commands (dark purple) and decoding algorithm (mustard) explicitly. During the decoding, the controller is required to execute stabilizer rounds for an unknown number of rounds, described by the **while** loop. That is, classical data is acquired during the decoding process. The uncertainty in the number of rounds gives rise to additional controlling parameters. For example, the decoder may return an error estimation or a Boolean output that indicates that the decoder's result is reliable, or alternatively, if the decoder performance variance is small, the user may predefine the number of stabilizer rounds after the measurement. In the pseudocode, we define the decoder to return two Boolean variables, one indicates the logical measurement result and the other indicates that the decoding was received. Stabilizer rounds are performed until the "**decoding_recieved**" variable turns to 1, and only thereafter the feed-forward is performed. Therefore, the benchmark definition for two decoding tasks checks aspects of the classical throughput, rather than only the latency. The experiment can run for further shots (expanding i to larger values) so that $FLR(n > 1)$ can be calculated and checked for convergence to 1 or not. If not, it is clear that the QECCP is insufficient for supporting fault-tolerant QEC with any calculation experiencing a divergent runtime.

The TIFL benchmark simplifies the requirements compared to those needed for non-Clifford gates, thanks to the distinct separation between decoding tasks. Such separation makes the TIFL benchmark executable with different decoding algorithms, which usually require well-defined boundaries (initialization and logical measurement). One decoding property that can be checked is the ability to start the decoding before the shot ends (i.e., before the data qubits are measured). For example, a lookup table decoder with a sliding window, as suggested in [42], can process syndromes and map them to physical bit-flips while the shot is still running. More scalable solutions that can enable low latencies include the recently suggested fusion-blossom decoder [44] or neural-network decoders [43]. In addition, to reduce the feed-forward latency, the decoder can potentially include another pre-decoding process in the form of parity checks between rounds (a.k.a detectors [47]) or advanced algorithms that can still run locally and reduce the syndromes that the decoder should analyse [57,58]. The benchmark is defined such that all these decoding algorithms can be checked as variants of the originally defined benchmark (defined with Union Find decoder) with near-term available quantum hardware to reach novel QEC experiments with real-time decoding.

Medium-term benchmark: Two Surgery Feed-forward Latency (TSFL)

As a second benchmark, we modify the TIFL benchmark so that the logical measurement will be a lattice surgery rather than a single surface measurement, as shown in Fig. 6 and defined in Listing 2. Such surgery feed-forward latency can be implemented in the near future and is one of the main building blocks for performing non-Clifford gates (also in color codes [59]). Moreover, as we showed above, the only type of feed-forward required for the entire quantum circuit comes as a result of a lattice surgery operation. Again, to realize the decoding throughput and the classical-quantum parallelization, we define the benchmark as a Two Surgery Feed-forward Latency (TSFL), from the moment one lattice surgery ends (where we define each surgery to be d stabilizer rounds) until the moment the second feed-forward is applied. To be explicit, we define the benchmark for a distance-3 or distance-5 configuration with a single surgery column (as in Fig. 1c), containing 41 or 111 physical qubits.

Figure 6. The Two Surgery Feed-forward Latency (TSFL) benchmark. (a) Logical circuit for measuring the TSFL benchmark, from the end of the first lattice surgery until the second feed-forward. The (circled) double line represents a (not-)conditional gate, and the red double arrows represent the feed-forward latencies and the TSFL benchmark. The stabilizer rounds are not shown since they represent a logical identity. **(b)** Surface view of the TSFL benchmark, showing that the boundary conditions of decoding task 1 will depend on decoding task 0 (as opposed to the TIFL benchmark).

```
rounds = 10
surgery_rounds = d
error_probability = 0.01
round_time = 1E-6
decoding_algorithm = Union_find
send_to_decoder(algorithm=decoding_algorithm)

initialize_surface(q0, state=0)
initialize_surface(q1, state=1)
for i in range(rounds):
    ancilla_bits = stabilizer_round(error_probability, round_time)
    send_to_decoder(ancilla_bits, task=0)
for i in [0,1]:
```

```
    initialize_surgery(q0, q1)
    for i in range(surgery_rounds):
        ancilla_bits = stabilizer_round(error_probability, round_time)
        send_to_decoder(ancilla_bits, task=i)
    surgery_data_bits = terminate_surgery((q0, q1), error_probability)
    if i == 0:
        tic = timestamp
    send_to_decoder(surgery_data_bits, task=i)
    [logical_result, decoding_recieved] = get_decoding_result(task=i)
    while not decoding_recieved:
        ancilla_bits = stabilizer_round(error_probability, round_time)
        send_to_decoder(ancilla_bits, task=i+1)
        [logical_result, decoding_received] = get_decoding_result(task=i)
    if xor(logical_result, i):
        play_x(q1)
    if i == 1:
        toc = timestamp

TSFT= toc - tic
```

Listing 2: Pseudocode for measuring the Two Surgery Feed-forward Latency (TSFL).
Color coding is the same as in Listing 1, the macros **initialize_surgery** and **terminate_surgery**
are defined in Listing S1.

Similar to the TILF benchmark, the TSFL benchmark also includes stabilizer rounds
until the feed-forward is applied (see Fig. 3c and Listing 2), so that the latency of the first
decoding task will determine the amount of processing data of the second decoding
task. However, a significant difference between the two benchmarks comes from the fact that
the decoding tasks are not well-separated. The second decoding task will have to remember
part of the syndromes from the first decoding task to overcome a set of errors with syndromes
in both tasks. This requirement significantly increases the challenge of decoding this
benchmark, and to our knowledge, it is currently impossible with the available decoding tools.
Hence, executing the TSFL benchmark will necessitate algorithmic development alongside
control hardware improvements.

V. Discussion

Above, we have explained the central roles of the QECCP, the necessity for a low feed-
forward latency, and suggested two benchmarks for evaluating the QECCP performance.
Although these benchmarks do not perform any useful quantum calculations, they are defined
using representative configurations according to current state-of-the-art parameters. Moreover,
they verify the ability of the QECCP to support future state-of-the-art QEC experiments, setting
the stage for scaling quantum hardware. These benchmarks, the commands within the
pseudocodes, and the operation regime analysis, are general for most other QEC stabilizer
codes which may have a different detailed physical pulse sequence and decoding algorithms.

17

The suggested benchmarks provide a first reduction of all QECCP components into a few benchmarking numbers. So far, each component has been evaluated separately. For example, quantum error correction codes were evaluated by their error threshold under optimal maximum-likelihood decoder [41](e.g., the XZZX surface code [60]), while decoding algorithms were benchmarked through their computational time, scalability, and error threshold. However, there is a lack of decoding latency benchmarks (only recently suggested in [44]) and a lack of integration benchmark that includes the quantum controller operation, the controller-decoder communication, and the connection to the actual quantum hardware with analogue signals. The proposed benchmarks provide the first steps for evaluating QECCP performance holistically.

In the long term, we expect a dominant role in parallelizing classical calculations in the QECCP. The controlling system will need to parallelize many decoding tasks to keep track of the frame of each logical qubit, and then merge decoding tasks during multi-surface lattice surgeries. Thus, it would be beneficial to consider indirect benchmarks for evaluating a QEC operation runtime, including the time it takes to run multiple shots, to compile logical circuits into T-gate operations [61], and to load the circuit parameters (e.g., to synthesize circuit waveform sequences). In this context, an interesting benchmark will be the time it takes to perform embedded calibrations for the circuit parameters optimization (e.g., [21–23]). The quality of the optimization will eventually determine the physical error probability of the circuit which in turn gives rise to another possible set of benchmarks related to the decoder adaptation time. The decoder will be required to adapt its weights and estimates of the circuit noise (for example by using optimal noise estimation [62]) and to adapt to catastrophic events [27,63].

We foresee that minimizing the benchmarks that we defined here while keeping high decoding accuracy will be at the core of quantum computation utility. Once the surgery latency is minimized, the effort will move towards minimizing the multi-surface surgery latency, and eventually the magic state distillation time. The distillation time is expected to determine de facto the whole quantum computation time for surface-code-based computation [48–50], which is currently the main expected route for reducing logical error rates. Finally, even if other promising QEC codes such as QLDPC codes will become dominant [64–67], all properties and blueprints for QECCP that we describe here will still stand. Thus, having clear benchmarks that combine the complete requirements of a QECCP is a vital and fundamental component in the development of quantum computing.

References

[1] P. W. Shor, *Fault-Tolerant Quantum Computation*, in *Proceedings of 37th Conference on Foundations of Computer Science* (IEEE Computer Society, Burlington, VT, USA, 1996), pp. 56–65.

[2] E. Knill, R. Laflamme, and W. H. Zurek, *Resilient Quantum Computation: Error Models and Thresholds*, Proc. R. Soc. Lond. Ser. Math. Phys. Eng. Sci. **454**, 365 (1998).

[3] R. P. Feynman, *Simulating Physics with Computers*, Int. J. Theor. Phys. **21**, 467 (1982).

[4] P. W. Shor, *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM J. Comput. **26**, 1484 (1997).

[5] D. Gottesman, Stabilizer Codes and Quantum Error Correction, Caltech, 1997.

[6] J. Preskill, *Reliable Quantum Computers*, Proc. R. Soc. Lond. Ser. Math. Phys. Eng. Sci. **454**, 385 (1998).

[7] E. T. Campbell, B. M. Terhal, and C. Vuillot, *Roads towards Fault-Tolerant Universal Quantum Computation*, Nature **549**, 172 (2017).

[8] L. Ella, L. Leandro, O. Wertheim, Y. Romach, R. Szmuk, Y. Knol, N. Ofek, I. Sivan, and Y. Cohen, *Quantum-Classical Processing and Benchmarking at the Pulse-Level*, arXiv:2303.03816.

[9] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, *Topological Quantum Memory*, J. Math. Phys. **43**, 4452 (2002).

[10] L. Riesebos, X. Fu, S. Varsamopoulos, C. G. Almudever, and K. Bertels, *Pauli Frames for Quantum Computer Architectures*, in *Proceedings of the 54th Annual Design Automation Conference 2017* (ACM, Austin TX USA, 2017), pp. 1–6.

[11] Google Quantum AI et al., *Suppressing Quantum Errors by Scaling a Surface Code Logical Qubit*, Nature **614**, 676 (2023).

[12] A. deMarti iOlius, P. Fuentes, R. Orús, P. M. Crespo, and J. E. Martinez, *Decoding Algorithms for Surface Codes*, arXiv:2307.14989.

[13] B. M. Terhal, *Quantum Error Correction for Quantum Memories*, Rev. Mod. Phys. **87**, 307 (2015).

[14] A. D. Corcoles, M. Takita, K. Inoue, S. Lekuch, Z. K. Minev, J. M. Chow, and J. M. Gambetta, *Exploiting Dynamic Quantum Circuits in a Quantum Algorithm with Superconducting Qubits*, Phys. Rev. Lett. **127**, 100501 (2021).

[15] T. Lubinski, C. Granade, A. Anderson, A. Geller, M. Roetteler, A. Petrenko, and B. Heim, *Advancing Hybrid Quantum–Classical Computation with Real-Time Execution*, Front. Phys. **10**, 940293 (2022).

[16] N. Ofek et al., *Extending the Lifetime of a Quantum Bit with Error Correction in Superconducting Circuits*, Nature **536**, 441 (2016).

[17] D. Devulapalli, E. Schoute, A. Bapat, A. M. Childs, and A. V. Gorshkov, *Quantum Routing with Teleportation*, arXiv:2204.04185.

[18] E. Bäumer, V. Tripathi, D. S. Wang, P. Rall, E. H. Chen, S. Majumder, A. Seif, and Z. K. Minev, *Efficient Long-Range Entanglement Using Dynamic Circuits*, arXiv:2308.13065.

[19] M. Foss-Feig et al., *Experimental Demonstration of the Advantage of Adaptive Quantum Circuits*, arXiv:2302.03029.

[20] C. Ryan-Anderson et al., *Realization of Real-Time Fault-Tolerant Quantum Error Correction*, Phys. Rev. X **11**, 041058 (2021).

[21] V. V. Sivak et al., *Real-Time Quantum Error Correction beyond Break-Even*, Nature **616**, 50 (2023).

[22] C. N. Barrett et al., *Learning-Based Calibration of Flux Crosstalk in Transmon Qubit Arrays*, Phys Rev Appl. **20**, 024070 (2023).

[23] P. V. Klimov et al., *Optimizing Quantum Gates towards the Scale of Logical Qubits*, arXiv:2308.02321.

[24] J. Kelly et al., *Scalable In-Situ Qubit Calibration during Repetitive Error Detection*, Phys. Rev. A **94**, 032321 (2016).

[25] Y. Xu, G. Huang, J. Balewski, A. Morvan, K. Nowrouzi, D. I. Santiago, R. K. Naik, B. Mitchell, and I. Siddiqi, *Automatic Qubit Characterization and Gate Optimization with* QubiC, ACM Trans. Quantum Comput. **4**, 1 (2023).

[26] A. Vepsäläinen et al., *Improving Qubit Coherence Using Closed-Loop Feedback*, Nat. Commun. **13**, 1932 (2022).

[27] M. McEwen et al., *Resolving Catastrophic Error Bursts from Cosmic Rays in Large Arrays of Superconducting Qubits*, Nat. Phys. **18**, 107 (2022).

[28] M. Cerezo et al., *Variational Quantum Algorithms*, Nat. Rev. Phys. **3**, 625 (2021).

[29] 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).

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

[31] S. Krinner et al., *Realizing Repeated Quantum Error Correction in a Distance-Three Surface Code*, Nature **605**, 669 (2022).

[32] D. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, *Surface Code Quantum Computing by Lattice Surgery*, New J. Phys. **14**, 123011 (2012).

[33] D. Litinski and F. von Oppen, *Lattice Surgery with a Twist: Simplifying Clifford Gates of Surface Codes*, Quantum **2**, 62 (2018).

[34] A. G. Fowler and C. Gidney, *Low Overhead Quantum Computation Using Lattice Surgery*, arXiv:1808.06709.

[35] C. Chamberland and E. T. Campbell, *Universal Quantum Computing with Twist-Free and Temporally Encoded Lattice Surgery*, PRX Quantum **3**, 010331 (2022).

[36] Y. Li, *A Magic State's Fidelity Can Be Superior to the Operations That Created It*, New J. Phys. **17**, 023037 (2015).

[37] R. S. Gupta et al., *Encoding a Magic State with beyond Break-Even Fidelity*, arXiv:2305.13581.

[38] C. Gidney, *Cleaner Magic States with Hook Injection*, arXiv:2302.12292.

[39] V. Kolmogorov, *Blossom V: A New Implementation of a Minimum Cost Perfect Matching Algorithm*, Math. Program. Comput. **1**, 43 (2009).

[40] N. Delfosse and N. H. Nickerson, *Almost-Linear Time Decoding Algorithm for Topological Codes*, Quantum **5**, 595 (2021).

[41] C. T. Chubb, *General Tensor Network Decoding of 2D Pauli Codes*, arXiv:2101.04125.

[42] P. Das, A. Locharla, and C. Jones, *LILLIPUT: A Lightweight Low-Latency Lookup-Table Based Decoder for Near-Term Quantum Error Correction*, arXiv:2108.06569.

[43] K. Meinerz, C.-Y. Park, and S. Trebst, *Scalable Neural Decoder for Topological Surface Codes*, Phys. Rev. Lett. **128**, 080505 (2022).

[44] Y. Wu and L. Zhong, *Fusion Blossom: Fast MWPM Decoders for QEC*, arXiv:2305.08307.

[45] O. Higgott and C. Gidney, *Sparse Blossom: Correcting a Million Errors per Core Second with Minimum-Weight Matching*, arXiv:2303.15933.

[46] F. Battistel, C. Chamberland, K. Johar, R. W. J. Overwater, F. Sebastiano, L. Skoric, Y. Ueno, and M. Usman, *Real-Time Decoding for Fault-Tolerant Quantum Computing: Progress, Challenges and Outlook*, arXiv:2303.00054.

[47] C. Gidney, *Stim: A Fast Stabilizer Circuit Simulator*, Quantum **5**, 497 (2021).

[48] J. Haah and M. B. Hastings, *Codes and Protocols for Distilling T, Controlled-S, and Toffoli Gates*, Quantum **2**, 71 (2018).

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

[50] M. E. Beverland, A. Kubica, and K. M. Svore, *The Cost of Universality: A Comparative Study of the Overhead of State Distillation and Code Switching with Color Codes*, PRX Quantum **2**, 020341 (2021).

[51] H. J. Briegel, D. E. Browne, W. Dür, R. Raussendorf, and M. V. den Nest, *Measurement-Based Quantum Computation*, Nat. Phys. **5**, 19 (2009).

[52] S. Bravyi and A. Kitaev, *Universal Quantum Computation with Ideal Clifford Gates and Noisy Ancillas*, Phys. Rev. A **71**, 022316 (2005).

[53] A. Erhard et al., *Entangling Logical Qubits with Lattice Surgery*, Nature **589**, 220 (2021).

[54] D. Herr, F. Nori, and S. J. Devitt, *Optimization of Lattice Surgery Is NP-Hard*, Npj Quantum Inf. **3**, 35 (2017).

[55] D. Gottesman, *The Heisenberg Representation of Quantum Computers*, arXiv:quant-ph/9807006.

[56] W. P. Livingston, M. S. Blok, E. Flurin, J. Dressel, A. N. Jordan, and I. Siddiqi, *Experimental Demonstration of Continuous Quantum Error Correction*, Nat. Commun. **13**, 2307 (2022).

[57] S. C. Smith, B. J. Brown, and S. D. Bartlett, *A Local Pre-Decoder to Reduce the Bandwidth and Latency of Quantum Error Correction*, Phys. Rev. Appl. **19**, 034050 (2023).

[58] L. Caune, J. Camps, B. Reid, and E. Campbell, *Belief Propagation as a Partial Decoder*, arXiv:2306.17142.

[59] A. J. Landahl and C. Ryan-Anderson, *Quantum Computing by Color-Code Lattice Surgery*, arXiv:1407.5103.

[60] J. P. B. Ataides, D. K. Tuckett, S. D. Bartlett, S. T. Flammia, and B. J. Brown, *The XZZX Surface Code*, Nat. Commun. **12**, 2172 (2021).

[61] L. Heyfron and E. T. Campbell, *An Efficient Quantum Compiler That Reduces T Count*, Quantum Sci. Technol. **4**, 015004 (2018).

[62] T. Wagner, H. Kampermann, D. Bruß, and M. Kliesch, *Optimal Noise Estimation from Syndrome Statistics of Quantum Codes*, Phys. Rev. Res. **3**, 013292 (2021).

[63] A. Siegel, A. Strikis, T. Flatters, and S. Benjamin, *Adaptive Surface Code for Quantum Error Correction in the Presence of Temporary or Permanent Defects*, Quantum **7**, 1065 (2023).

[64] N. Delfosse, M. E. Beverland, and M. A. Tremblay, *Bounds on Stabilizer Measurement Circuits and Obstructions to Local Implementations of Quantum LDPC Codes*, arXiv:2109.14599.

[65] C. A. Pattison, A. Krishna, and J. Preskill, *Hierarchical Memories: Simulating Quantum LDPC Codes with Local Gates*, arXiv:2303.04798.

[66] Q. Xu, J. P. B. Ataides, C. A. Pattison, N. Raveendran, D. Bluvstein, J. Wurtz, B. Vasic, M. D. Lukin, L. Jiang, and H. Zhou, *Constant-Overhead Fault-Tolerant Quantum Computation with Reconfigurable Atom Arrays*, arXiv:2308.08648.

[67] S. Gu, E. Tang, L. Caha, S. H. Choe, Z. He, and A. Kubica, *Single-Shot Decoding of Good Quantum LDPC Codes*, arXiv:2306.12470.

Acknowledgement

Control Requirements and Benchmarks for Quantum Error Correction

Yaniv Kurman[1], Lior Ella[1], Ramon Szmuk[1], Oded Wertheim[1], Benedikt Dorschner[2], Sam Stanwyck[2], and Yonatan Cohen[1]

[1]*Quantum Machines Inc., Tel Aviv, Israel*

[2]*NVIDIA Corp, Santa Clara, CA, USA*

S1 - The necessity of a feed-forward correction of a Pauli frame flip

We present here a short example that shows a case where a Pauli frame flip propagates to a non-Clifford flip after two non-Cifford gates. Without the loss of generality, let us consider the following example, where we want to execute two non-commuting non-Clifford gates: $T = diag(1, e^{i\pi/4})$ followed by $X^{1/4}$ (as in Fig. 3a), where the qubit experiences a Pauli X error before the T gate. After the T gate, the X frame is converted to a Clifford correction in the form of $TXT^{\dagger} = (X + Y)/\sqrt{2}$. Then, after the $X^{1/4}$ gate, the correction is converted to a non-Clifford gate (a π rotation around the vector $(\frac{1}{\sqrt{2}}, \frac{1}{2}, \frac{1}{2})$) which cannot propagate efficiently in software throughout the quantum circuit. Therefore, from this example, we can deduce that a detection of a Pauli frame changes must be corrected through a circuit modification (or an active pulse) before the Pauli correction becomes a non-Clifford correction. Specifically, if the error is detected during the decoding of the T gate, it should be corrected before the execution of the $X^{1/4}$ gate.

S2 - How additional ancilla surfaces can ease the latency requirements

As we explain in section III of the main text, and in section S1, fault-tolerant useful quantum calculations require feedback where the feed-forward type can change according to the specific non-Clifford gate (as in Fig. 1b). However, there is a scheme, namely the auto-corrected $\pi/8$ scheme [S1], where the feed-forward logical gate remains independent of the executed non-Clifford gate. Moreover, this scheme can scale and ease the latency requirements with enough ancillary surfaces. The scheme can execute any Pauli (and multi-qubit Pauli) $\pi/8$ gate, as shown in Fig. S1 for a single T gate followed by a $X^{1/4}$ gate with two ancillary logical qubits for each non-Clifford gate.

Conceptually, the QECCP role does not change compared to the example in Fig 1 of the main text when executing a non-Clifford gate fault-tolerantly. That is, the execution of a planned gate includes an ancillary surface initialization stage, a lattice surgery stage, a logical measurement stage, and a feed-forward. The lattice surgery is performed between the surface that is initialized in the $|T\rangle = |0\rangle + e^{i\pi/4}|1\rangle$ state, the computational logical qubit (or qubits), and the $|0\rangle$ ancillary surface, producing two measurement results m_0 and m_1. As in the example in the main text, it is only the outcome from the surgery with the magic state (m_1) that determines the feed-forward conditional operation that will be in the form of the measurement basis of the second ancillary surface, i.e., whether to apply a Hadamard gate to the data qubits of this surface just before they are measured. To successfully execute the planned non-Clifford gate, along with the subsequent non-Clifford gates, all four measurements, including those of the two ancilla surfaces, are required to determine the Pauli frame update for the computational surfaces.

Figure S1. Performing non-Clifford gates with the auto-corrected $\pi/8$ scheme. (a) An example of the auto-corrected $\pi/8$ for performing a single T gate followed by a $X^{1/4}$ gate with 2 ancilla surfaces. The scheme includes three stages (ancilla initialization, cyan; lattice surgery, blue; and measurement, purple), where the classical outcome of one of the surgeries (m_1) will determine the feed-forward in the form of a logical Hadamard gate to the second ancillary surface just before it is measured. Therefore, the second ancillary surface exhibits stabilizer rounds until the surgery decoding ends. This feed-forward (and also state initialization) is similar to every $\pi/8$ gate. **(b)** The surface view of the circuit in (a), with different decoding tasks in different colors. The threshold latency L_{th} is determined by the number of rounds needed for a fault-tolerant logical measurement. **(c)** An implementation of a similar logical circuit with four ancillary surfaces. The additional surfaces eliminate any gap between the two non-Clifford gates, since the feed-forward is directed to an ancillary surface rather than the computational one. **(d)** The surface view of the circuit in (c), showing how four ancilla surfaces reduce the computation time. Through this view, it becomes evident that another contribution to the reduction in computation time is achieved by decreasing the data analysed by the decoder (yellow decoding task), thereby reducing the circuit delay (δ_1).

The auto-corrected $\pi/8$ scheme has additional advantages when allowing more than two ancillary surfaces. For example, when having four ancillary surfaces (as shown in Fig. S1c), the additional surfaces enable performing two non-Clifford gates without any gaps in time between the gates. This is enabled because the feed-forward is applied to an ancillary surface rather than the computational surface. This capability yields another advantage: as the computational surfaces don't require waiting, the decoder processes a reduced number of syndromes (depicted in yellow area in Fig. S1d compared to yellow area in S1b), resulting in an overall reduced feed-forward latency. Therefore, not only the second non-Clifford gate can start before the feed-forward of the first non-Clifford gate is applied, the additional ancillary surfaces enable a smaller latency, and thus a reduced logical clock. A similar effect will happen when six ancillary surfaces are available (Fig. S2a), where the difference compared to four surfaces is that three consecutive nonstop non-Clifford gates can be applied (and further nonstop gates if $L < L_{th}$). Only when the number of ancillary surfaces increases to eight (Fig. S2b), L_{th} starts to increase without increasing the decoding task area. Fig. S3 presents the effect of increasing the threshold latency on the decoding latencies for various classical configurations. As L_{th} increases, the requirements on the classical computation are relieved.

Figure S2. The $\pi/8$ architecture for 6 (a) and 8 (b) ancilla surfaces. In both panels, the computational surfaces ($|\psi\rangle$) are continuously involved in a lattice surgery, i.e., without any stabilizer rounds where the computational surfaces are not involved in lattice surgery. Each main decoding task is encircled in a different color. Since there are three basic operations (surface initialization, lattice surgery, and logical measurement) at least eight ancillary surfaces are needed to increase L_{th}.

Figure S3: The effect of the threshold latency on the decoding latency. The decoding latency for different threshold latencies and classical parameters (denoted in panel (c)). If L_{th} is increased beyond L^0, fault-tolerant quantum computation can be executed even when using a seemingly impractical classical decoder (large linear utilization U_{lin} and complexity α).

S3 - Analysis of the linear decoder behavior

In this analysis we address the case where the feed-forward latency grows linearly with the number of data it receives, $L(N) = \tau_0 + N/T$, with τ_0 being a latency offset and T is the decoder's throughput. As we explained in section III in the main text, if there is a round where $L^{(n)} > L_{th}$ (for simplicity let's denote the latency of this round as L_0), then the additional data in the following decoding task becomes $N^{(1)}(L_0) = N_0 + \lambda(L_0 - L_{th})$. We note that the for surface code architecture, the syndrome arrival rate to the decoder can be expressed as $\lambda = \frac{pM_c d^2}{T_s}$ where M_c is the number of computational surfaces, d is the code distance, T_s is the time of a stabilizer rounds, and p is the probability that a measurement is a syndrome defect. Thus, the additional data of $N^{(1)}$ compared to N_0 arises from stabilizer rounds of the computational surfaces that wait for the next lattice surgery. Due to this additional data, the consequent feed-forward latency becomes

$$L^{(1)} = \tau_0 + \frac{N^{(1)}}{T} = \tau_0 + \frac{N_0 + \lambda(L_0 - L_{th})}{T} = L^0 + \frac{\lambda}{T}(L_0 - L_{th}) = L_0 + U(L_0 - L_{th}) \quad (S1)$$

where U is the utilization of the decoder. Similarly, the latency of the next non-Clifford gate will be

$$L^{(2)} = L_0 + U(L^{(1)} - L_{th}) = L^{(1)} + U^2(L_0 - L_{th}) = L_0 + U(U + 1)(L_0 - L_{th}),$$

which we can continue recursively to the latency of the n'th non-Clifford gate,

$$L^{(n)} = L^{(n-1)} + U^n(L_0 - L_{th}) = L_0 + (L_0 - L_{th})\sum_{k=1}^{n} U^k$$

$$= L_0 + (L_0 - L_{th})\frac{U(U^n - 1)}{U - 1}. \quad (S2)$$

From Eq. (S2) we can see that the latency diverges with n if the decoder's utilization satisfies $U > 1$. This condition will occur if syndrome arrival rate to the decoder is larger than the the decoder's throughput.

```
func initialize_surface(logical_qubit, state):
        active_logical_qubits.append(logical_qubit)
        for data_qubit in logical_qubit.data_qubits:
                reset(data_qubit)
                if state:
                        play(pi,data_qubit)

func play_x(logical_qubit):
        for data_qubit in logical_qubit.data_qubits:
                play(pi,data_qubit)

func measure_surface(logical_qubit,error_probability):
        active_logical_qubits - = logical_qubit
        return measure_qubits(logical_qubit.data_qubit_resonators, error_probability)

func stabilizer_round(error_probability, round_time):
        for logical_qubit in active_logical_qubits:
                ancilla_bits=measure_qubits(logical_qubit.ancilla_qubit_resonators,
error_probability)
        wait_until(round_time)
        return ancilla_bits

func measure_qubits(qubit_resonators, error_probability):
        discr_threshold=threshold(error_probability)
        for i, resonator in enumerate(qubit_resonators):
                measure(readout_pulse, resonator, demod(x))
                state[i] = x > discr_threshold
        return state

func initialize_surgery(logical_qubit1,logical_qubit2):
        active_logical_qubits - = logical_qubit1
        active_logical_qubits - = logical_qubit2
        active_logical_qubits.append(logical_qubit1+logical_qubit2+sugregy_qubits)
        for data_qubits in sugregy_qubits.data_qubits:
                reset(data_qubit)

func terminate_surgery((logical_qubit1,logical_qubit2),error_probability)
        measure_qubits(sugregy_qubits.data_qubits, error_probability))
        active_logical_qubits - =(logical_qubit1+logical_qubit2+sugregy_qubits)
        active_logical_qubits.append(logical_qubit1)
        active_logical_qubits.append(logical_qubit2)
```

Listing S1: Pseudocode for the macros in the benchmark definitions. Color coding is the same as in Listing 1 in the main text. The parameter **discr_threshold** is the discrimination threshold, chosen such that white noise in the analog channels will have a probability according to the **error_probability** that can probabilistically create an error for a specific qubit resonator, given the measurement history (denoted by **meas_history** which includes all measurement results so far) to be above it. We define the benchmarks for a 7-to-1 readout multiplexing so that a Surface-49 will have 7 pairs of analog channels for all measurements. We note also that the command **reset** can be defined as a macro with iterations in quantum-real-time to reach a high fidelity.

Table S1: Simulation parameters

Figure	4a (purple)	4b	4c	4d	S3
code distance d	5				
syndrome probability p	0.01				
# computational surfaces M_c	2				
number of initial syndromes N_0	$3pM_cd^3$				
latency offset coefficient τ_0		3			
latency linear unitization U_{lin}		variable			[0.6, 0.9, 1.1, 1.5]
latency prefactor τ_1		U_{lin}/pM_cd^2			
latency complexity factor α		variable		1	[0.9, 1, 1.1]
Initial latency L^0		$\tau_0 + \tau_1(N_0)^\alpha$			
threshold latency L_{th}	11				[11, 15, 25]

Supplementary references

[S1] D. Litinski, *A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery*, Quantum **3**, 128 (2019).