Introduction

Quantum computers currently face significant hurdles regarding noise and decoherence, which threaten the stability of quantum information. For the QuEra challenge, we were tasked with mitigating these effects by "catching" decohered qubits, with the ultimate goal of improving the performance of a logical qubit functioning as a memory. To address this, we implemented a comprehensive Quantum Error Correction (QEC) protocol on TSim and parallelized our circuit. In this paper, we discuss how we achieved the following:

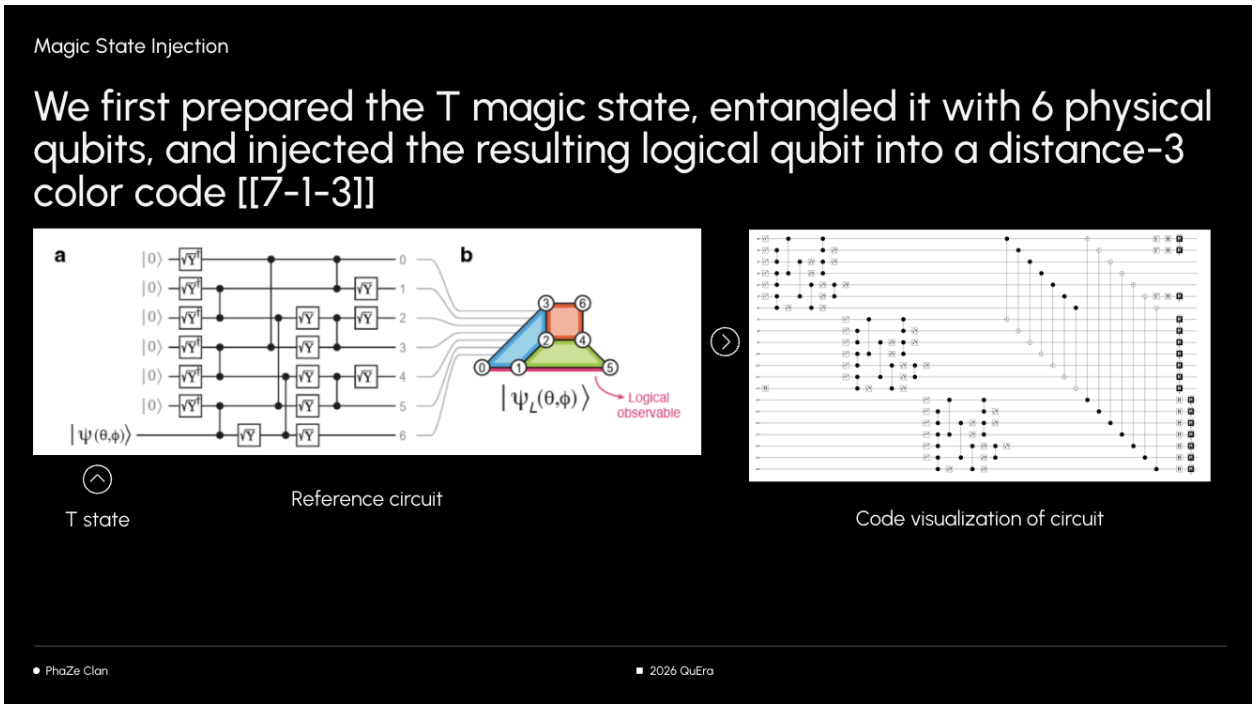
- **Magic State Injection:** Magic states are quantum states that enable universal quantum computation, along with the Clifford operations (which are Hadamard, phase, CNOT, and Pauli measurements). Clifford gates can only be reliably implemented in quantum hardware, so the magic states are prepared and teleported using Clifford operations and measurements. The magic states are usually encoded and error-corrected. We first prepared a magic state (T state) and injected it onto a distance-3 color code.
- **Error Correction:** We utilized Steane's error correction process via ancilla qubits to identify and capture unintentional bit flips (X) and phase flips (Z) resulting from environmental noise.
- **Non-Magic State Injection:** We injected a non-magic state (zero state) into a [[7-1-3]] color code to evaluate and compare improvements in logical error rates with the magic state in the same color code.
- **Noise Model Analysis:** We designed four custom noise channels to group related noise types and observe the success of our QEC process. We graphed logical error as a function of the global scale of physical error, observing distinct power laws and performance breakdowns across different initialized states and QEC processes.
- **Scaling Up With [[17-1-5]] (Bonus 1):** We experimented with increasing the code distance to 5 (using a 17-1-5 color code) to evaluate error rate suppression at larger system sizes.
- **Benchmarking (Distance-3 vs Distance-5):** We observed and explained differences in fidelity rates across distance-3 vs distance-5 code.
- **Active Correction (Bonus 2 and 4) Attempt:** Although we understood the feedforward logic necessary to demonstrate quantum memory via syndrome decoding, time constraints prevented us from successfully implementing the classical control required to trigger the corrective quantum gates.

Magic State Injection

To initialize our error-corrected memory, we first focused on the generation and encoding of a logical qubit using a [[7-1-3]] color code. Our approach relied on distributing quantum information across multiple physical qubits to protect it from local noise.

We began by preparing a magic T-state and injecting it onto a distance-3 color code. This process utilized the specific circuit layout shown in the image below, designed to map the state

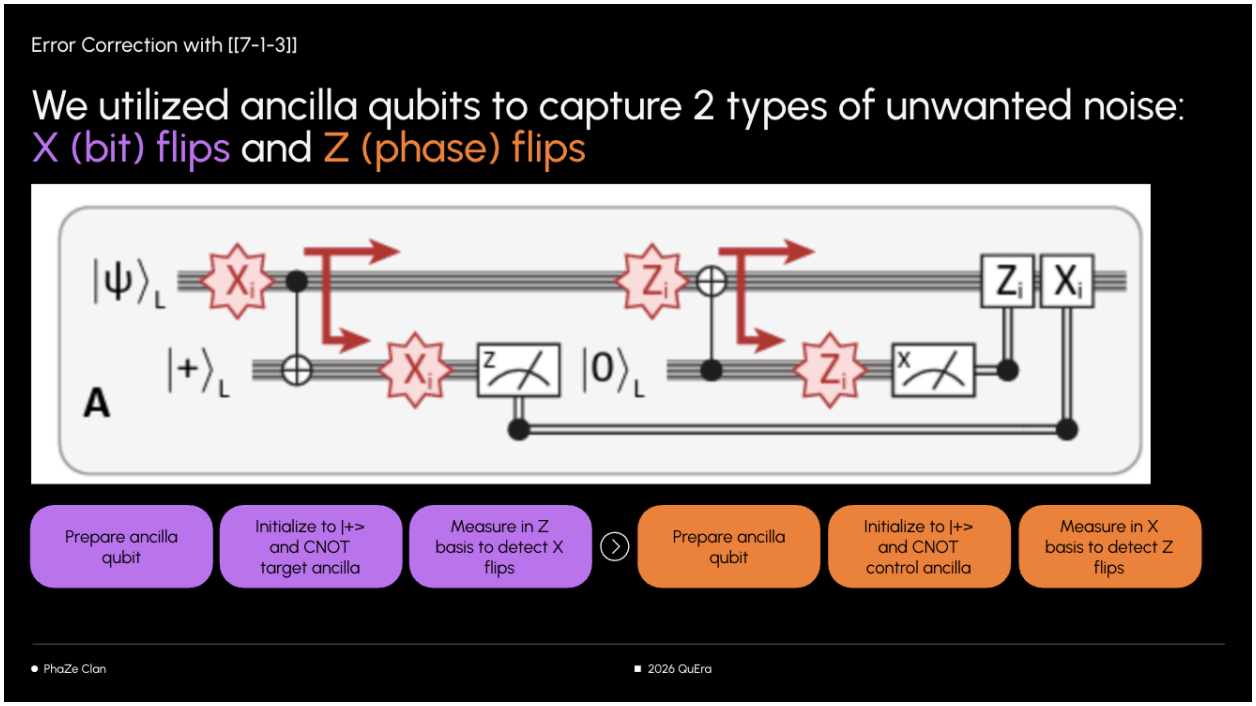
onto the code space. This also entangles the initial magic state with 6 additional physical qubits. This operation "shares" the quantum information across all 7 physical qubits, effectively unifying them to act as a single, robust logical qubit.



Error Correction

Ancilla Initialization

To protect the logical qubit, we implemented a Steane error correction protocol using ancilla qubits, each mirroring the structure of our logical qubit. This setup allowed us to detect and filter out the two primary types of noise: bit flips (X errors) and phase flips (Z errors). Please see below for more detail on this process.



- **Capturing X Flips:** The first ancilla was initialized with the $|+\rangle$ state. We performed CNOT gates with the ancilla acting as the target, followed by a measurement in the Z basis (the default basis).
- **Capturing Z Flips:** The second ancilla was initialized with the $|0\rangle$ state. We performed CNOT gates with the logical qubit acting as the target, followed by a measurement in the X basis (implemented via a Hadamard gate followed by a Z measurement).

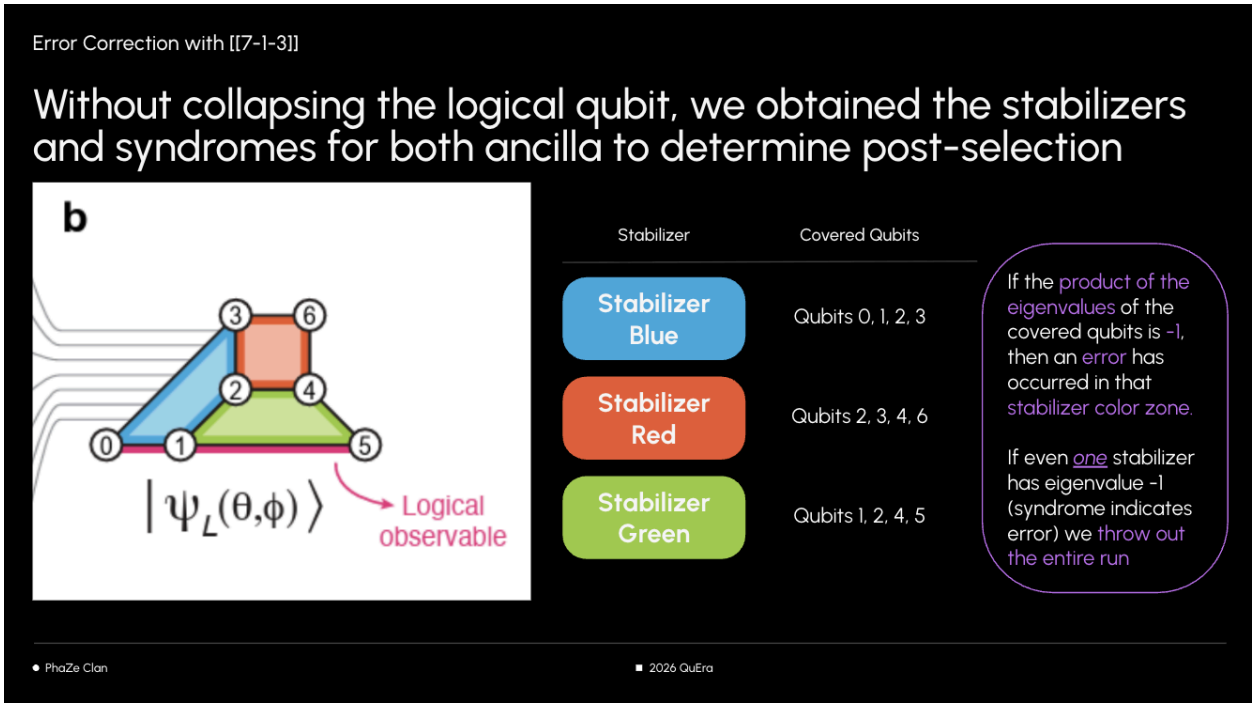
Stabilizer Measurement and Syndrome Extraction

The geometry of the color code dictates the stabilizer measurements. Each color in the lattice corresponds to a group of four physical qubits:

- **Blue:** Qubits 0, 1, 2, 3
- **Red:** Qubits 2, 3, 4, 6
- **Green:** Qubits 1, 2, 4, 5

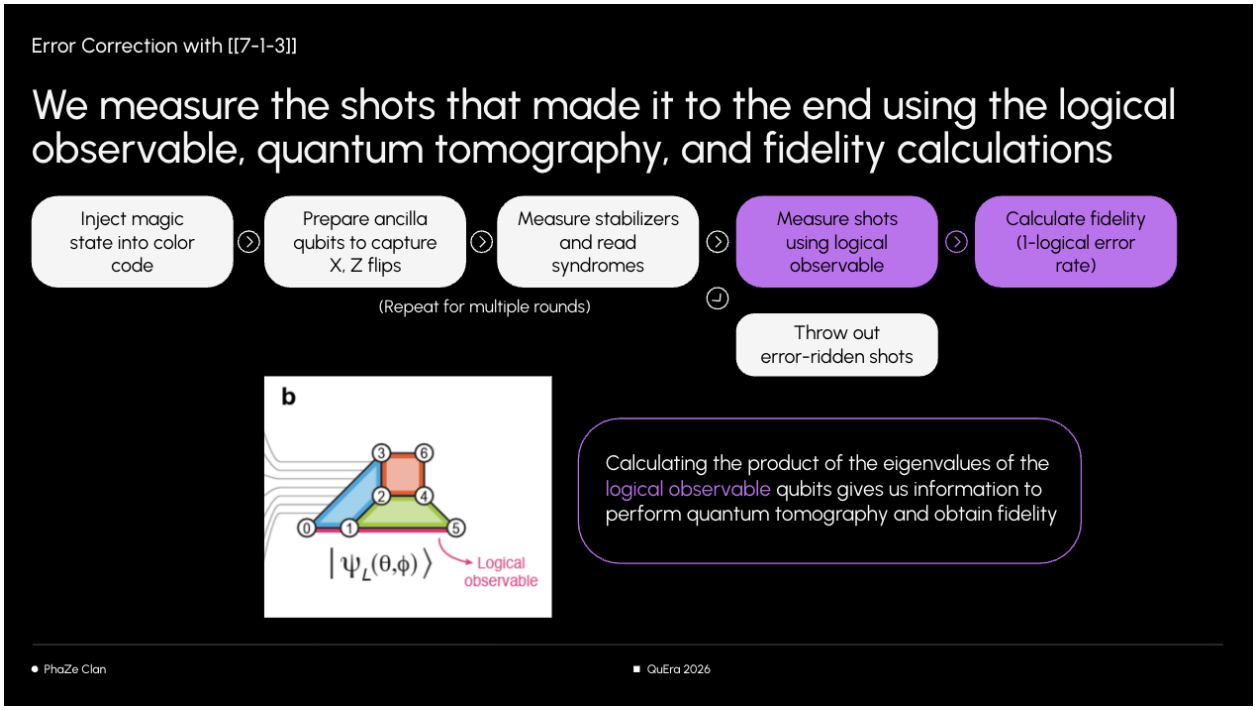
We calculated the stabilizers for each color group, defined as the product of the eigenvalues of the associated physical qubits. A stabilizer value of 1 indicates no error, while -1 signals an error. We measured 3 stabilizers for each error type (X and Z), resulting in a total of 6 stabilizer values across 2 syndromes per shot. By measuring the stabilizers of specific qubit subsets defined by the distance-3 color code lattice, we could identify when the logical state had been corrupted without collapsing the stored quantum information.

Afterwards, we utilized a post-selection strategy. If any stabilizer returned a -1 (indicating either an X or Z error), we discarded the shot entirely to prevent contaminating the final results with noisy data (see detailed post-selection process below).



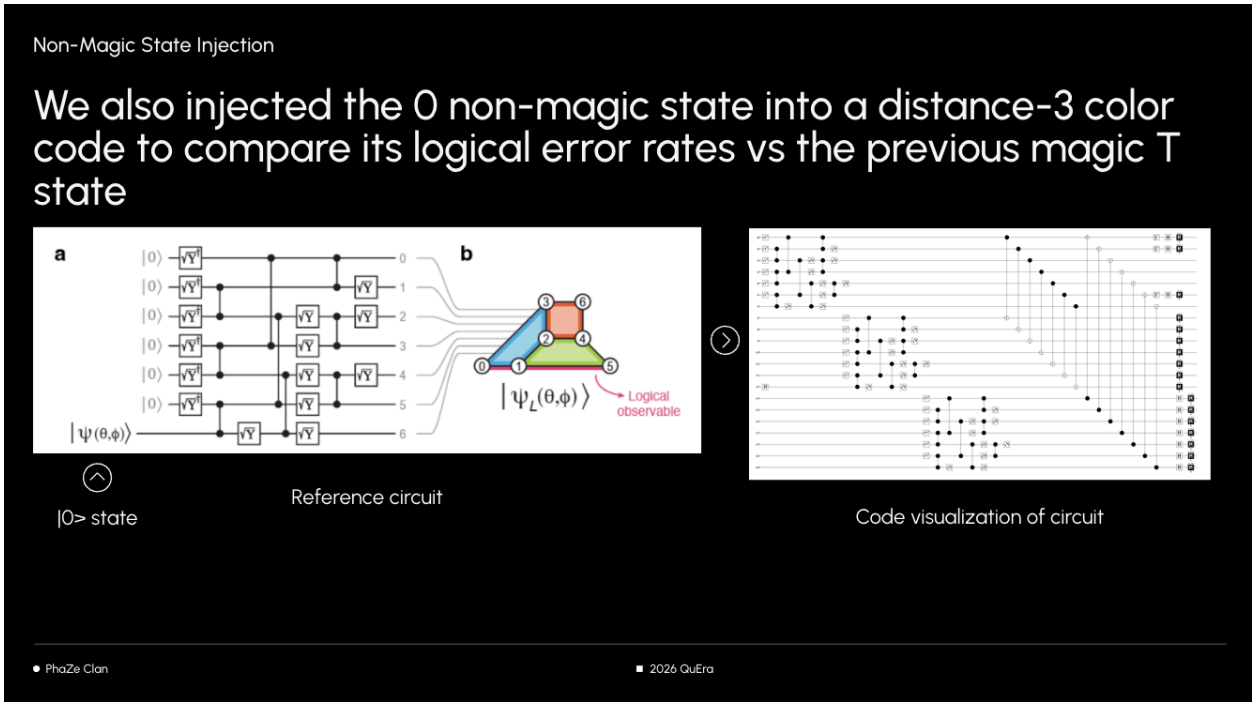
Logical Observable, Quantum Tomography, and Fidelity

On the remaining valid shots that were not thrown out from post-selection, we measured the logical observable (the product of the eigenvalues of physical qubits 0, 1, and 5). This data was later used to construct a density matrix via quantum tomography, allowing us to calculate the fidelity, a metric from 0 to 1 representing how closely the final noisy qubit matched the original ideal state (see below for a visual explanation of the role of the logical observable and our overall QEC process).



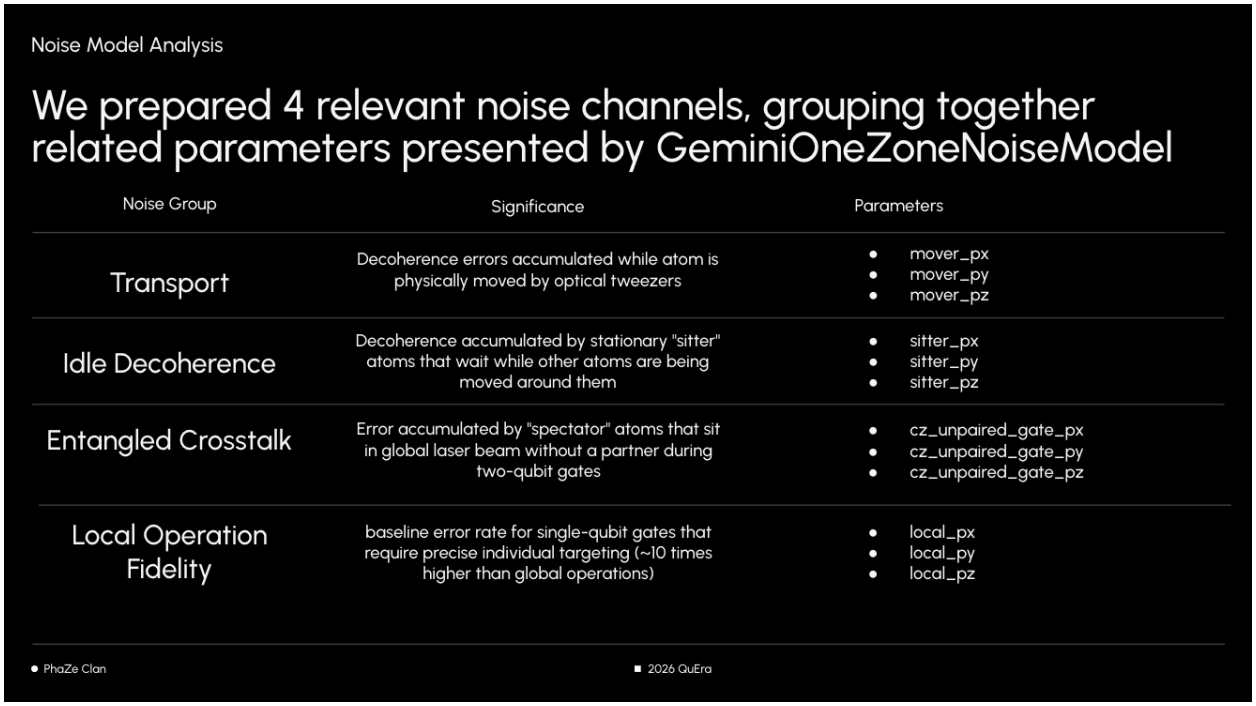
Non-Magic State Injection

To isolate the contribution of the error-correction protocol itself from state-dependent effects, we also injected a non-magic logical state (|0>) and evaluated its logical error rate under the same four noise models and physical error-rate sweeps, using the identical QEC pipeline described in the previous *Error Correction* section.



Noise Model Analysis

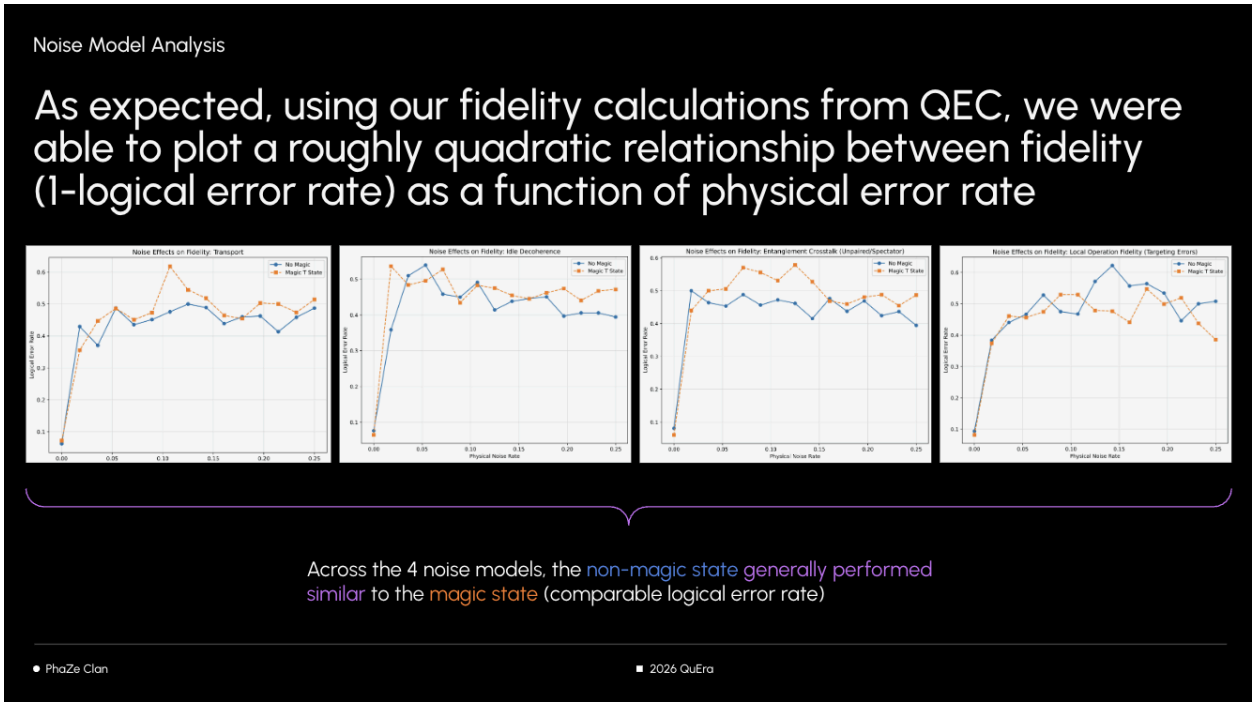
To evaluate the fault-tolerance performance of the [[7,1,3]] quantum error-correcting code under realistic hardware conditions, we constructed four distinct noise models, each corresponding to a physically motivated grouping of related parameters provided by the GeminiOneZoneNoiseModel. These models isolate different prevalent error mechanisms (i.e., transport, idle decoherence, entangled crosstalk, and local operation fidelity), allowing us to probe how specific noise sources impact logical performance (see below for a table of chosen noise models, descriptions, and associated parameters).



For each noise model, physical error rates were swept across a specified range (0-0.25), and circuit execution was repeated for a fixed number of shots to obtain statistically meaningful estimates of logical error rates, which is derived by subtracting fidelity from 1.

Magic State vs Non-Magic State Results

Across the explored noise regimes, we observed that the logical error rates for magic and non-magic injections were largely comparable, with no clear or systematic advantage for either case (see below for results).

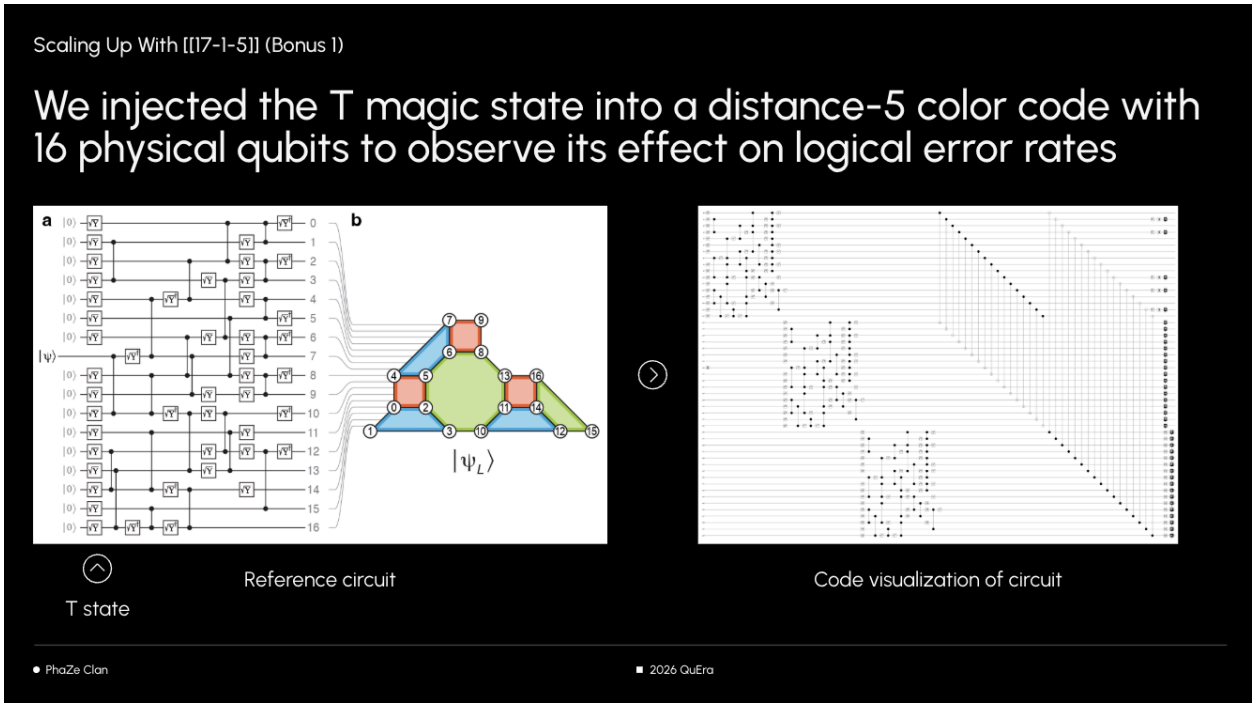


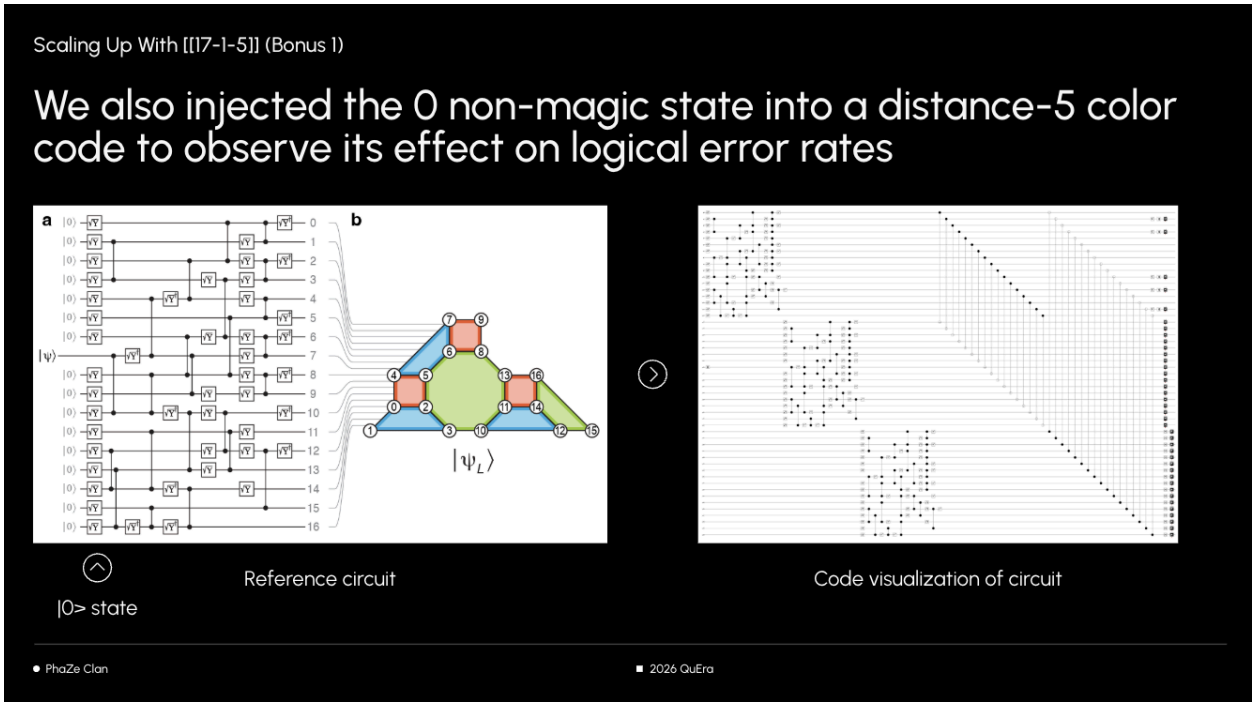
This behavior is consistent with expectations in a Clifford-only simulation backend such as TSim, where noise is modeled stochastically and state evolution is tracked through stabilizer structure rather than full quantum amplitudes. In this setting, the dominant contributors to logical failure are circuit depth, fault propagation during encoding, and syndrome-extraction errors, all of which affect magic and non-magic states similarly at the logical level. Since both injected states are ultimately subjected to the same non–fault-tolerant encoding circuit and identical stabilizer measurements, the logical error rate is governed more by where and how faults occur in the circuit than by whether the initial state is magic. Differences between magic and non-magic states are therefore expected to become significant only when coherent phase information, non-Clifford error channels, or subsequent magic-state–specific processing (e.g., distillation or gate teleportation) are explicitly modeled, effects that are outside the representational scope of TSim. As a result, the observed similarity in performance reflects a limitation of the simulation regime rather than an indication that magic and non-magic states are fundamentally equivalent under fault-tolerant encoding.

For each sampled physical error rate p, we computed the logical error rate (1 - fidelity) of the final logical state with respect to the target injected logical state (magic or non-magic). Plotting the logical error rate against the physical error rate reveals the characteristic power-law behavior expected from quantum error correction. In our noise range, the curves exhibit an approximately quadratic scaling, consistent with theoretical expectations for a distance-3 code.

Scaling Up With [[17-1-5]] (Bonus 1)

To explore scaling behavior beyond the distance-3 regime, we increased the code distance to 5 by implementing a [[17,1,5]] color code, as shown in the image below, and repeated the analysis for both magic (T) and non-magic (|0>) injected states. As with the [[7,1,3]] code, logical information is extracted from stabilizer eigenvalues formed by parity checks over color-grouped qubits, but the distance-5 construction introduces additional stabilizers, larger color groups, and deeper syndrome-extraction circuits.



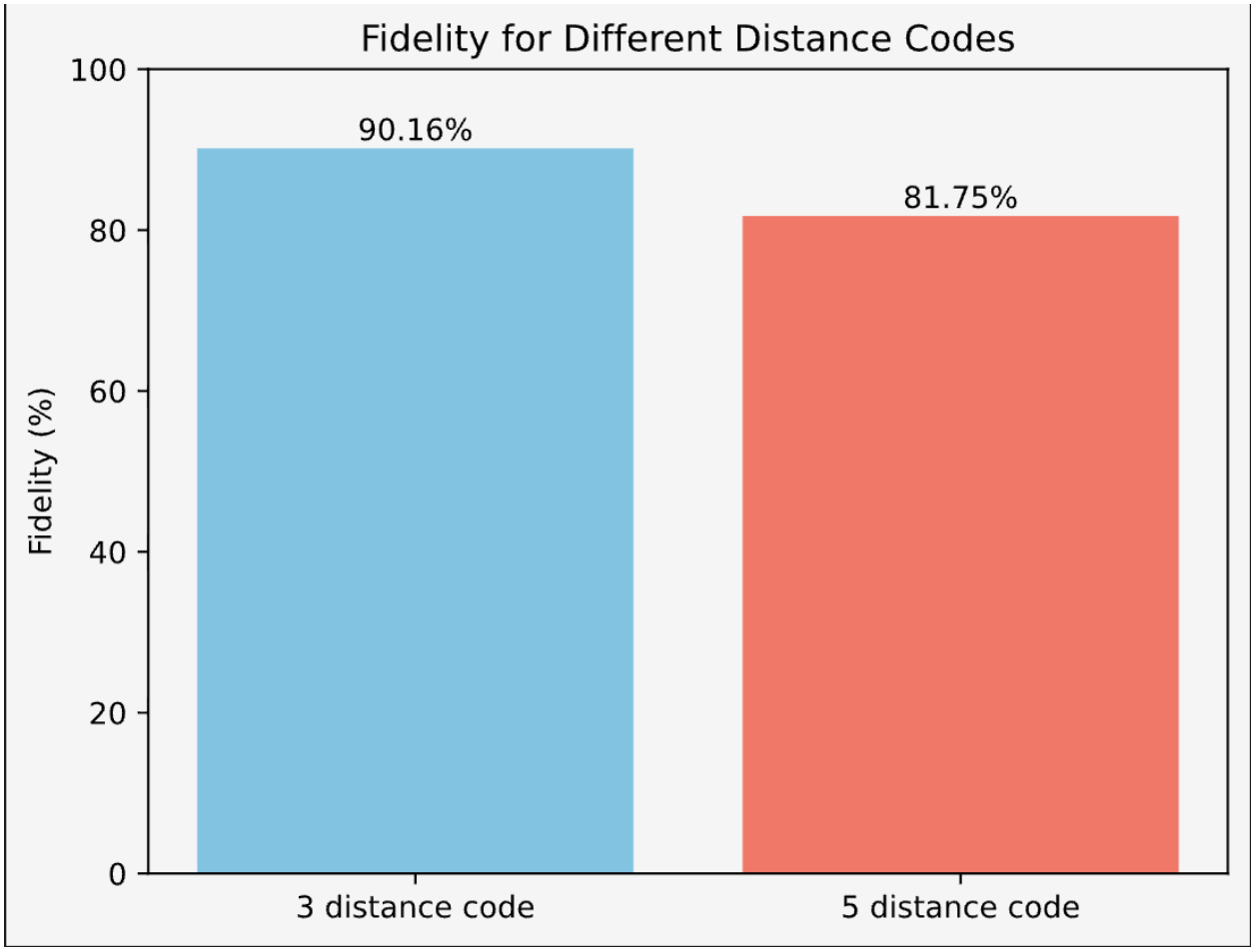


Benchmarking (Distance-3 vs Distance-5)

We benchmarked performance on the 4 noise models between 4 combinations of state initialization and QEC protocol:

- Magic state injected into [[7-1-3]] color code
- Non-magic state injected into [[7-1-3]] color code
- Magic state injected into [[17-1-5]] color code
- Non-magic state injected into [[17-1-5]] color code

Although we initially planned to compare logical error rates across all four noise models, time constraints led us to prioritize distance-3 and distance-5 fidelity comparisons to highlight the effects of the more complex initial state preparation in the distance-5 code.

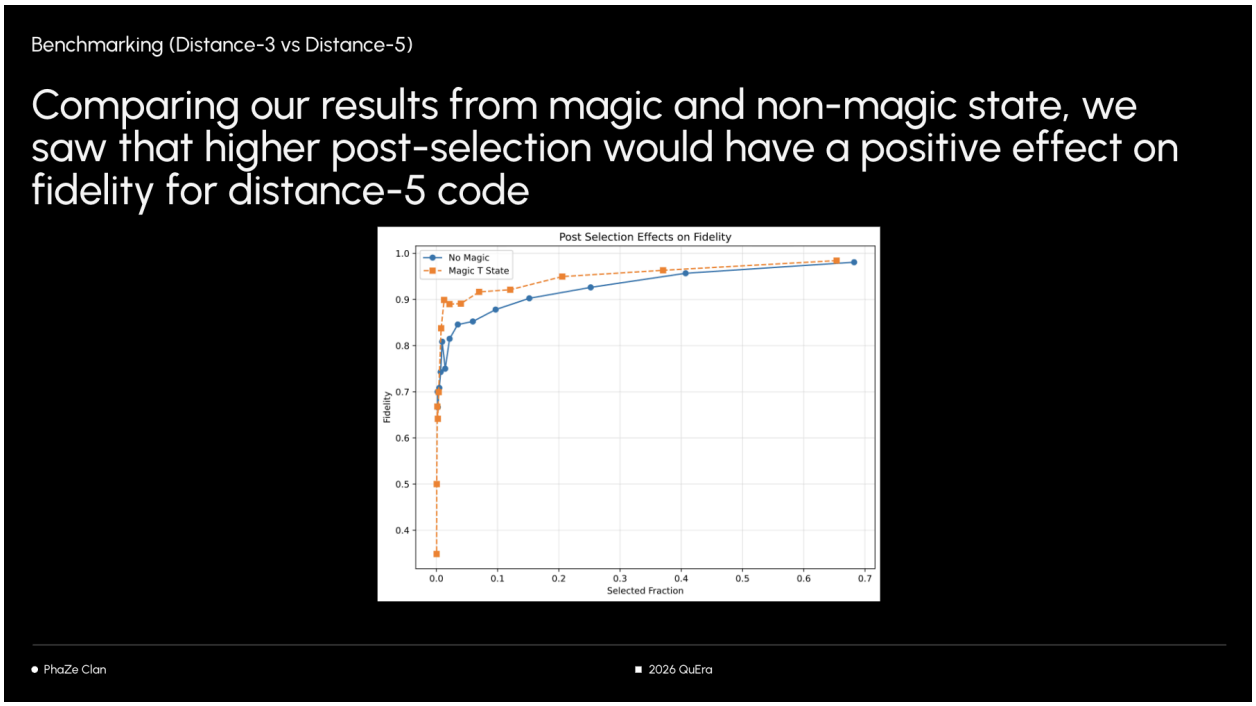


The lower fidelity observed at distance-5 arises primarily because the encoding circuit is significantly more complex and non-fault-tolerant. Scaling to distance-5 requires manipulating many more physical qubits with many more gate operations than smaller codes; this increased complexity causes more errors to accumulate during the initial state preparation, directly reducing the "injected fidelity." Additionally, because the experimental gate fidelities have not yet surpassed the circuit threshold, the larger code cannot efficiently compensate for this added noise without aggressive postselection, making the performance drop more pronounced than in the simpler distance-3 case.

Contribution of Post-Selection

Across both magic and non-magic states for distance-5 code, we observed that a higher and more aggressive post-selection rate did indeed have the expected positive effect on fidelity, due to its role as a highly-selective filter. Specifically, this technique mitigates the risk of error propagation: a critical issue where a fault in a single qubit does not remain isolated, but is instead transferred to coupled qubits during multi-qubit operations, thereby amplifying a localized noise event into a systemic corruption of the global state. While post-selection cannot physically prevent an error on a control qubit from spreading to a target qubit during an entangling gate, post-selection allows us to identify the source of the error after the fact. By

discarding the run based on the control qubit's failure, we essentially excise the propagated error from the final dataset, ensuring that the remaining shots reflect only the uncorrupted interactions.



Active Correction (Bonus 2 and 4) Attempt

The task was to decode the qubit using the syndromes and recover the original information. This conveys the idea of memory because we would be able to persist the qubit past its decoherence time by applying a correction on the physical qubit based on information from stabilizers and syndromes. On the implementation side, we were having trouble using classical gates to control when quantum gates, such as the X and Z, are used. We understand the feedforward logic, and if we had some more time, we likely would have been able to implement this feature.

```
@kernel
def circuit():
    # Allocate qubits for logical, plus, and zero blocks
    q_logical = squin.qalloc(7)
    q_plus = squin.qalloc(7)
    q_zero = squin.qalloc(7)

    # Inject magic state if requested
    if inject_magic:
        squin.broadcast.h(q_logical[-1])
        squin.broadcast.t_adj(q_logical[-1])

    # Prepare auxiliary + state
    for _ in range(n_iterations):
        squin.broadcast.h(q_plus[-1])
        encode_block(q_logical)
        encode_block(q_plus)
        encode_block(q_zero)

        # Entangle logical, plus, and zero blocks
        squin.broadcast.cx(q_logical, q_plus)
        squin.broadcast.cx(q_zero, q_logical)

    # Measure blocks
    measure_all(q_plus)
    squin.broadcast.h(q_zero)
    measure_all(q_zero)

    # Classical logic involving if statements
    # <===== Here ======>

    measure_subset(q_logical)

return circuit
```

```python
def get_bit_and_stabilizers(kernel):
    """
    Run the kernel and extract logical measurement outcomes and stabilizers.
    """
    # Sample measurement outcomes
    samples = kernel.compile_sampler(seed=0).sample(shots=5000, batch_size=10000)

    # Split samples into logical qubits, plus block, and zero block
    logical_combination = samples[:, :3]
    plus_syndromes = samples[:, 3:10]
    zero_syndromes = samples[:, 10:17]

    # Compute logical bit values
    logical_bit = eigen_calc_prod(logical_combination)

    # Compute X-type stabilizers from + block
    plus_stabilizers = np.stack([
        eigen_calc_prod(plus_syndromes[:, [0, 1, 2, 3]]),
        eigen_calc_prod(plus_syndromes[:, [1, 2, 4, 5]]),
        eigen_calc_prod(plus_syndromes[:, [2, 4, 6, 3]])
    ], axis=1)

    # Compute Z-type stabilizers from 0 block
    zero_stabilizers = np.stack([
        eigen_calc_prod(zero_syndromes[:, [0, 1, 2, 3]]),
        eigen_calc_prod(zero_syndromes[:, [1, 2, 4, 5]]),
        eigen_calc_prod(zero_syndromes[:, [2, 4, 6, 3]])
    ], axis=1)

    # Combine X and Z stabilizers
    stabilizers = np.concatenate([plus_stabilizers, zero_stabilizers], axis=1)

    return logical_bit, stabilizers
```

We would have to move the classical logic in code screenshot 2 into the circuit in screenshot 1.