

Companion Compendium

Encoded Magic-State Preparation
on the $[[4, 2, 2]]$ Code

A Gentle Introduction to the Physics, Engineering,
and Optimisation Behind the Notebook Series

Companion to the AUTORESEARCH-QUANTUM project

<https://github.com/saymrwulf/autoresearch-quantum>

April 2026

This compendium is the “course textbook” for the eight Jupyter notebooks in the AUTORESEARCH-QUANTUM project. It is designed to be read before, during, or after working through the notebooks. Every concept exercised in the notebooks is explained here with the depth and context that a tutorial session cannot provide. No prior knowledge of quantum error correction is assumed; familiarity with linear algebra and complex numbers is helpful.

Contents

1	Why This Project Exists	6
1.1	The Promise of Quantum Computing	6
1.2	The Noise Problem	6
1.3	Quantum Error Correction: The Path Forward	6
1.4	The Role of Magic States	7
1.5	What This Project Does	7
2	Qubits, Gates, and Circuits	8
2.1	The Qubit	8
2.1.1	The Bloch Sphere	8
2.1.2	Global Phase	8
2.2	Quantum Gates	9
2.2.1	Single-Qubit Gates	9
2.2.2	Two-Qubit Gates	9
2.3	Circuits	9
3	The Magic State	10
3.1	Definition	10
3.2	Why Magic States Matter	10
3.3	Three Ways to Prepare $ T\rangle$	11
4	The $[[4, 2, 2]]$ Error-Detecting Code	12
4.1	Why Encode?	12
4.2	Code Parameters	12
4.3	The Codespace	13
4.3.1	Properties of the Stabilisers	13
4.4	Logical Operators	13
4.4.1	The Two Logical Qubits	14
4.5	The Encoder Circuit	14
4.5.1	The Encoded State	14
4.6	Error Detection	14
5	Measurement, Verification, and Postselection	16
5.1	The Measurement Problem	16

5.2	Ancilla-Based Syndrome Extraction	16
5.3	Postselection	16
5.4	Witness Circuits	17
6	The Magic-State Witness	18
6.1	What Is a Witness?	18
6.2	The Formula	18
6.2.1	The Magic Factor	18
6.2.2	The Spectator Factor	18
6.2.3	Ideal Witness Value	19
6.3	Witness vs. Fidelity	19
7	Noise and the Hardware Reality	20
7.1	Sources of Noise	20
7.2	Noise Models and Simulators	20
7.3	Transpilation: From Logical to Physical	20
7.4	Cost Model	21
8	Scoring: Putting It All Together	22
8.1	The Weighted Acceptance-Cost Score	22
8.2	Factory Throughput Score	22
8.3	Failure Modes	23
9	The Ratchet: Learning by Doing	24
9.1	The Incumbent-Challenger Model	24
9.1.1	The Bootstrap Incumbent	24
9.2	Challenger Generation Strategies	24
9.2.1	NeighborWalk (40% of budget)	24
9.2.2	RandomCombo (30% of budget)	25
9.2.3	LessonGuided (30% of budget)	25
9.3	Ratchet Steps and Rungs	25
9.4	Lesson Extraction	25
9.5	Search Space Narrowing	26
9.6	Cross-Rung Propagation	26
9.7	Transfer Evaluation	26
10	The Five Rungs	27
11	Putting It All Together: The Pipeline	28
12	Glossary	30
A	Mathematical Background	32
A.1	Complex Numbers and Amplitudes	32
A.2	Tensor Products	32

A.3	Pauli Matrices	32
A.4	Eigenvalues and Expectation Values	33
B	Notebook–Compendium Cross-Reference	34

Chapter 1

Why This Project Exists

1.1 The Promise of Quantum Computing

A quantum computer manipulates quantum bits—*qubits*—that can exist in superpositions of $|0\rangle$ and $|1\rangle$. Where a classical bit is either 0 or 1, a qubit is described by

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle, \quad |\alpha|^2 + |\beta|^2 = 1, \quad (1.1)$$

where α and β are complex numbers called *amplitudes*. The constraint $|\alpha|^2 + |\beta|^2 = 1$ ensures that the probabilities of measuring $|0\rangle$ or $|1\rangle$ sum to one.

Multiple qubits can be *entangled*, meaning the state of one depends on the state of the others in a way that has no classical analogue. Entanglement is the engine that gives quantum algorithms their power: Shor’s algorithm for factoring, Grover’s algorithm for search, and quantum simulation of molecules all exploit entanglement.

1.2 The Noise Problem

Today’s quantum processors are *noisy*. Every gate operation, every idle moment, and every measurement introduces errors. A typical two-qubit gate on current IBM hardware has an error rate of roughly 1%, which sounds small but compounds rapidly: a circuit of 100 two-qubit gates has only about a 37% chance of executing perfectly ($0.99^{100} \approx 0.37$).

This is the central engineering challenge of our era in quantum computing: the algorithms we want to run require thousands of gates, but the hardware can barely manage a few hundred before errors dominate.

1.3 Quantum Error Correction: The Path Forward

The solution is *quantum error correction* (QEC). The idea is conceptually simple: spread the information of one *logical qubit* across several *physical qubits*, in such a way that errors can be detected and corrected without disturbing the encoded information.

This is analogous to classical error-correcting codes (like the parity checks in your hard drive), but with a quantum twist: you cannot copy a qubit (the *no-cloning theorem*), and measurement generally destroys the state. Quantum codes must work around both constraints.

1.4 The Role of Magic States

Even with error correction, there is a fundamental limitation. The *Eastin–Knill theorem* states:

The Eastin–Knill Theorem

No quantum error-correcting code admits a universal set of *transversal* gates. That is, you cannot implement every gate you need by simply applying the same operation to each physical qubit independently.

Most codes can implement Clifford gates (Hadamard H , phase gate S , CNOT) transversally, but Cliffords alone are not enough. The *Gottesman–Knill theorem* proves that any circuit built entirely from Clifford gates can be efficiently simulated on a classical computer—no quantum advantage.

To break out of this trap, you need a *non-Clifford resource*. The simplest and most common is the T gate (a $\pi/8$ rotation). Rather than applying T directly on the encoded qubits (which would not be transversal), the standard approach is:

1. Prepare a special auxiliary state called the **magic state** $|T\rangle$.
2. Consume $|T\rangle$ via *gate teleportation* to apply the T gate to an encoded qubit.

This compendium—and the entire AUTORESEARCH-QUANTUM project—is about step 1: preparing magic states as reliably and efficiently as possible.

1.5 What This Project Does

The project builds a complete pipeline for encoded magic-state preparation on the $[[4, 2, 2]]$ quantum error-detecting code:

1. **Prepare** the magic state $|T\rangle$ using different gate sequences (seed styles).
2. **Encode** it into the $[[4, 2, 2]]$ code.
3. **Verify** the encoding using stabiliser measurements.
4. **Measure** the quality using a magic-state witness.
5. **Score** the experiment balancing quality, acceptance rate, and cost.
6. **Optimise** the parameters automatically using a ratchet that learns from its own results.

The notebooks let you see, interact with, and modify every step.

Chapter 2

Qubits, Gates, and Circuits

2.1 The Qubit

A single qubit lives in a two-dimensional complex vector space $\mathcal{H} = \mathbb{C}^2$, with the *computational basis* $\{|0\rangle, |1\rangle\}$. Any pure state is

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}, \quad |\alpha|^2 + |\beta|^2 = 1. \quad (2.1)$$

2.1.1 The Bloch Sphere

Every single-qubit state can be visualised as a point on the *Bloch sphere*. Writing $|\psi\rangle = \cos(\theta/2) |0\rangle + e^{i\phi} \sin(\theta/2) |1\rangle$, the state maps to the point $(\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \theta)$ on a unit sphere.

- $|0\rangle$ is at the north pole $(0, 0, 1)$.
- $|1\rangle$ is at the south pole $(0, 0, -1)$.
- States on the equator ($\theta = \pi/2$) have equal probability of being measured as $|0\rangle$ or $|1\rangle$.

2.1.2 Global Phase

If you multiply the entire state by $e^{i\gamma}$, you get a new vector $e^{i\gamma} |\psi\rangle$ that *cannot be distinguished* from $|\psi\rangle$ by any measurement. This factor is called the *global phase*, and it is physically irrelevant.

Subtlety

Two states that differ only by a global phase are the same physical state. Their fidelity is 1.0, and they occupy the same point on the Bloch sphere. In the notebooks, you will see three seed styles that produce different-looking amplitude vectors but fidelity 1.0—this is why.

2.2 Quantum Gates

Quantum gates are *unitary* transformations: operations that preserve the norm of the state vector. Every gate U satisfies $U^\dagger U = I$.

2.2.1 Single-Qubit Gates

The most important single-qubit gates:

Gate	Matrix	What it does
Pauli X	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	Bit-flip: $ 0\rangle \leftrightarrow 1\rangle$
Pauli Z	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	Phase-flip: $ 1\rangle \mapsto - 1\rangle$
Pauli Y	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	Both bit-flip and phase-flip: $Y = iXZ$
Hadamard H	$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	Creates superposition: $ 0\rangle \mapsto +\rangle$
Phase S	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	Quarter-turn around Z
T gate	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$	Eighth-turn around Z : the key non-Clifford gate

The gates $\{H, S, \text{CNOT}\}$ generate the *Clifford group*. Adding T promotes the set to a *universal* gate set: any unitary can be approximated to arbitrary precision.

2.2.2 Two-Qubit Gates

The most important two-qubit gate is the **CNOT** (controlled-NOT, also called **CX**):

$$\text{CNOT} = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \quad (2.2)$$

It flips the target qubit if and only if the control qubit is $|1\rangle$. CNOT is the primary entangling gate and the dominant source of noise on current hardware.

Intuition

On IBM hardware, two-qubit gates have error rates 10–100× higher than single-qubit gates. Minimising the two-qubit gate count is the single most impactful optimisation for circuit quality.

2.3 Circuits

A quantum circuit is a sequence of gates applied to a register of qubits. Time flows left to right. Qubits are drawn as horizontal lines (“wires”). Each gate is a box or symbol on its wire(s).

After the gates, you *measure* some or all qubits, collapsing their superposition into classical bits. A *shot* is one execution of the full circuit (preparation + gates + measurement).

Chapter 3

The Magic State

3.1 Definition

The magic state is defined as:

$$|T\rangle = \frac{|0\rangle + e^{i\pi/4}|1\rangle}{\sqrt{2}}. \quad (3.1)$$

T-state Properties

- Amplitudes: $\alpha = 1/\sqrt{2}$, $\beta = e^{i\pi/4}/\sqrt{2} = (1+i)/(2)$.
- Phase of $|1\rangle$ coefficient: $\pi/4 = 45^\circ$.
- Bloch sphere: on the equator ($\langle Z \rangle = 0$), at 45° between the $+X$ and $+Y$ axes.
- $\langle X \rangle = \langle Y \rangle = 1/\sqrt{2} \approx 0.7071$.

$\pi/4$ vs. $\pi/8$

The gate is called “ T ” and sometimes the “ $\pi/8$ gate” because of Bloch-sphere conventions (the rotation *angle* is $\pi/4$, but the *half-angle* in the Bloch parametrisation is $\pi/8$). The *state* has phase $\pi/4$. The notebooks use $\pi/4$ consistently.

3.2 Why Magic States Matter

As discussed in chapter 1, the Clifford group alone is classically simulable (Gottesman–Knill theorem). The T gate breaks this barrier. But implementing T transversally on most error-correcting codes is impossible (Eastin–Knill). The workaround:

1. Prepare $|T\rangle$ in an auxiliary register.
2. Use *gate teleportation*: a circuit of Clifford gates plus a measurement that effectively applies T to the target qubit, consuming $|T\rangle$ in the process.
3. If the $|T\rangle$ is noisy, apply *magic-state distillation* to purify it (at the cost of more copies).

Our project focuses on step 1: preparing the highest-quality $|T\rangle$ we can, encoded in an error-detecting code, so that downstream distillation (if needed) starts from the best possible input.

3.3 Three Ways to Prepare $|T\rangle$

On a single qubit, the magic state can be prepared by several equivalent gate sequences. We call these *seed styles*:

Style	Gates	Notes
<code>h_p</code>	H then $P(\pi/4)$	Most natural: Hadamard creates $ +\rangle$, phase gate adds $\pi/4$
<code>ry_rz</code>	$R_Y(\pi/2)$ then $R_Z(\pi/4)$	Native on many hardware platforms
<code>u_magic</code>	$U(\pi/2, \pi/4, 0)$	Single parameterised gate

All three produce the same physical state (fidelity = 1.0). The amplitude vectors may look different because they differ by a *global phase*—which, as we discussed, is unphysical. The choice of seed style matters only when the circuit is *transpiled* for a specific hardware backend, because different decompositions lead to different native-gate counts and thus different noise profiles.

Chapter 4

The $[[4, 2, 2]]$ Error-Detecting Code

4.1 Why Encode?

A bare qubit has no protection against errors. If a cosmic ray or a stray photon flips a qubit, the computation is silently corrupted. We need a way to detect (and ideally correct) such errors.

Classical error detection is straightforward: store redundant copies and compare them. But the *no-cloning theorem* forbids copying an unknown quantum state:

Theorem 4.1 (No-Cloning). There is no unitary operation U such that $U |\psi\rangle |0\rangle = |\psi\rangle |\psi\rangle$ for all $|\psi\rangle$.

Quantum error correction circumvents this by encoding information not in copies but in *entanglement patterns*. The information is spread across multiple physical qubits in a way that individual errors can be detected without revealing the encoded data.

4.2 Code Parameters

The $[[4, 2, 2]]$ code is the smallest quantum error-*detecting* code. Its parameters mean:

Parameter	Meaning
$n = 4$	4 physical qubits
$k = 2$	2 logical qubits encoded
$d = 2$	Distance 2: detects any single-qubit error

Detection vs. Correction

Distance $d = 2$ means the code can *detect* any error affecting a single qubit, but it *cannot correct* it. If an error is detected, the shot is discarded (postselection). A code needs distance $d \geq 3$ to correct single-qubit errors.

For our purposes, detection is sufficient: we discard corrupted shots and keep only clean ones.

This trades quantity for quality—a deliberate choice that the scoring formula captures.

4.3 The Codespace

The 4 physical qubits span a Hilbert space of dimension $2^4 = 16$. The $[[4, 2, 2]]$ code selects a 4-dimensional subspace called the *codespace* \mathcal{C} . Within \mathcal{C} , the two logical qubits can be in any state—giving us $2^2 = 4$ degrees of freedom, as expected.

The codespace is defined by two *stabiliser* operators:

Definition 4.1 (Stabilisers of the $[[4, 2, 2]]$ code) $S_X = X \otimes X \otimes X \otimes X = XXXX$, $S_Z = Z \otimes Z \otimes Z \otimes Z = ZZZZ$. (4.1) A state $|\psi\rangle$ is in the codespace if and only if $S_X |\psi\rangle = +|\psi\rangle$ and $S_Z |\psi\rangle = +|\psi\rangle$.

In other words, the codespace is the simultaneous $+1$ eigenspace of both stabilisers. Any state outside this eigenspace has been corrupted by an error.

4.3.1 Properties of the Stabilisers

Both stabilisers have important algebraic properties:

1. **Squaring to identity:** $S_X^2 = I$ and $S_Z^2 = I$. Since $S^2 = I$, the eigenvalues of S can only be ± 1 .
2. **Commutation:** $[S_X, S_Z] = S_X S_Z - S_Z S_X = 0$. The two stabilisers commute, so they can be measured simultaneously (they share a common eigenbasis).
3. **Hermiticity:** Both are Hermitian ($S^\dagger = S$), so they are valid observables.

4.4 Logical Operators

Within the codespace, we need operators that act on the *logical* qubits without leaving the codespace. These must commute with both stabilisers.

For our encoded magic state, the relevant logical operators are:

Operator	Pauli string	Qubits acted on	Role
Logical X	$IXIX$	0, 2	X on the magic logical qubit
Logical Y	$IXZY$	0, 1, 2	Y on the magic logical qubit
Spectator Z	$ZIZI$	1, 3	Z on the spectator logical qubit

Intuition

Why does logical Y act on 3 physical qubits? Because the logical information is *distributed* across all physical qubits by the encoding. Logical operators must act on this distributed encoding. There is no single “logical qubit wire” to put a Y gate on.

4.4.1 The Two Logical Qubits

The $[[4, 2, 2]]$ code encodes two logical qubits:

1. **Logical qubit 0 (“the magic qubit”)**: Prepared in the magic state $|T\rangle$. We measure $\langle X_L \rangle$ and $\langle Y_L \rangle$ to assess its quality.
2. **Logical qubit 1 (“the spectator”)**: Prepared in $|0\rangle_L$. We measure $\langle Z_{\text{spec}} \rangle$ to confirm it has not been disturbed. Ideally $\langle Z_{\text{spec}} \rangle = +1$.

4.5 The Encoder Circuit

The encoder is a unitary circuit that maps a product state on 4 qubits into an entangled codeword:

$$|\psi\rangle_L = U_{\text{enc}}(|T\rangle_0 \otimes |0\rangle_1 \otimes |0\rangle_2 \otimes |0\rangle_3). \quad (4.2)$$

The project implements two encoder styles:

cx_chain (5 CNOT gates, depth 7): A cascade of CNOT gates that entangles all four qubits. The Hadamard on qubit 3 creates the necessary superposition for the second logical qubit (the spectator). This is the default encoder.

cz_compiled (5 CZ gates, depth 11): Uses controlled-Z (CZ) gates instead of CNOT. CZ is the native two-qubit gate on some hardware platforms (e.g. Google’s processors). Although the depth is higher, transpilation may produce fewer native gates on CZ-native hardware.

Both encoders produce the same logical state (fidelity = 1.0). The choice between them is an *engineering* decision: which decomposition produces fewer errors after transpilation for a specific backend?

4.5.1 The Encoded State

After encoding with the default **cx_chain**, the magic state has 4 non-zero amplitudes out of 16 possible basis states:

$$|T\rangle_L = \frac{1}{2} |0000\rangle + \frac{e^{i\pi/4}}{2} |0101\rangle + \frac{e^{i\pi/4}}{2} |1010\rangle + \frac{1}{2} |1111\rangle. \quad (4.3)$$

The pattern $\{0000, 0101, 1010, 1111\}$ is the codespace of the $[[4, 2, 2]]$ code. Any measurement outcome outside these four bitstrings indicates an error.

4.6 Error Detection

The power of the code lies in its ability to detect errors. Consider what happens when a single-qubit Pauli error acts on the encoded state:

Error type	$\langle XXXX \rangle$	$\langle ZZZZ \rangle$	Detected by
No error	+1	+1	—
X on any qubit	+1	−1	$ZZZZ$
Z on any qubit	−1	+1	$XXXX$
Y on any qubit	−1	−1	Both

Why $ZZZZ$ Detects X Errors

$ZZZZ$ detects X errors because X and Z *anti-commute*: $XZ = -ZX$. When an X error occurs on one qubit, it flips the sign of the $ZZZZ$ eigenvalue from +1 to −1.

Similarly, $XXXX$ detects Z errors. A Y error ($Y = iXZ$) triggers both stabilisers.

Every single-qubit error flips at least one stabiliser—this is the *distance-2 guarantee*. A weight-2 error (two qubits affected simultaneously) could go undetected; that is the inherent limitation of distance 2.

Chapter 5

Measurement, Verification, and Postselection

5.1 The Measurement Problem

To check whether an error has occurred, we need to measure the stabilisers $\langle XXXX \rangle$ and $\langle ZZZZ \rangle$. But there is a problem: *directly measuring the data qubits collapses the superposition and destroys the encoded state.*

5.2 Ancilla-Based Syndrome Extraction

The solution is to use *ancilla qubits*—extra qubits that are entangled with the data qubits in a controlled way, then measured. The measurement outcome of the ancilla (called the *syndrome*) tells us whether an error occurred, without revealing the actual data.

Intuition

Think of the ancilla as a thermometer: it reads the “temperature” of the data qubits (error or no error) without disturbing the patient (the encoded state). The syndrome bits are the thermometer’s reading.

In the $[[4, 2, 2]]$ code, we extract two syndrome bits:

- Syndrome bit for $ZZZZ$: +1 (no X error) or -1 (X error detected).
- Syndrome bit for $XXXX$: +1 (no Z error) or -1 (Z error detected).

We encode these as bits: 0 means +1 (no error), 1 means -1 (error detected). The syndrome string “00” means both stabilisers are satisfied—the state is in the codespace.

5.3 Postselection

Postselection is the process of keeping only the shots where the syndrome indicates no error:

Definition 5.1 (Postselection). Given N total shots, let N_{accept} be the number with syndrome “00”. The *acceptance rate* is

$$r_{\text{accept}} = \frac{N_{\text{accept}}}{N}. \quad (5.1)$$

All other shots are discarded.

The Cost of Postselection

Postselection improves quality by filtering out corrupted shots. But it *reduces the number of usable data points*. If the acceptance rate is 60%, you need $\sim 1.7\times$ as many total shots to get the same statistical power. The scoring formula accounts for this trade-off.

On an ideal (noiseless) simulator, the acceptance rate is 100%—every shot passes the syndrome check. On a noisy backend, some shots will trigger the syndrome flag and be discarded. The acceptance rate is therefore a direct measure of how much noise affects the experiment.

5.4 Witness Circuits

After postselection, we need to measure the logical operators to assess the quality of the encoded magic state. This requires three separate circuits:

1. **Logical X circuit:** Measures $\langle X_L \rangle$ by measuring qubits 0 and 2 in the X basis.
2. **Logical Y circuit:** Measures $\langle Y_L \rangle$ by measuring qubits 0, 1, 2 in the appropriate bases.
3. **Spectator Z circuit:** Measures $\langle Z_{\text{spec}} \rangle$ by measuring qubits 1 and 3 in the Z basis.

Why Three Separate Circuits?

The three logical operators *do not commute* with each other. Measuring one would disturb the quantum state in a way that invalidates the measurement of the others. Therefore, each operator requires its own copy of the experiment.

Chapter 6

The Magic-State Witness

6.1 What Is a Witness?

A *witness* is a number computed from measurement results that quantifies how close the prepared state is to the ideal target. Unlike fidelity (which requires full state tomography), the witness uses only a few expectation values and can be estimated efficiently.

6.2 The Formula

The magic-state witness for the $\llbracket 4, 2, 2 \rrbracket$ encoded $|T\rangle$ is:

$$W = \underbrace{\frac{1 + \frac{\langle X_L \rangle + \langle Y_L \rangle}{\sqrt{2}}}{2}}_{\text{magic factor}} \times \underbrace{\frac{1 + \langle Z_{\text{spec}} \rangle}{2}}_{\text{spectator factor}} \quad (6.1)$$

6.2.1 The Magic Factor

The magic factor measures how well the encoded state matches the T -state character. For the ideal $|T\rangle$:

- $\langle X_L \rangle = 1/\sqrt{2} \approx 0.7071$
- $\langle Y_L \rangle = 1/\sqrt{2} \approx 0.7071$
- Magic factor $= (1 + (0.7071 + 0.7071)/\sqrt{2})/2 = (1 + 1)/2 = 1.0$

6.2.2 The Spectator Factor

The spectator factor checks that the second logical qubit (which should be in $|0\rangle_L$) has not been disturbed:

- Ideal: $\langle Z_{\text{spec}} \rangle = +1$
- Spectator factor $= (1 + 1)/2 = 1.0$

6.2.3 Ideal Witness Value

For a perfect preparation:

$$W_{\text{ideal}} = 1.0 \times 1.0 = 1.0. \quad (6.2)$$

Any noise or error reduces W below 1. The witness is deliberately *sensitive*: even moderate noise produces a noticeable drop, making it a useful diagnostic.

Witness Under Noise

Suppose noise reduces the logical expectations to $\langle X_L \rangle = 0.55$, $\langle Y_L \rangle = 0.50$, and $\langle Z_{\text{spec}} \rangle = 0.02$:

$$\text{Magic factor} = \frac{1 + (0.55 + 0.50)/\sqrt{2}}{2} = \frac{1 + 0.742}{2} = 0.871, \quad (6.3)$$

$$\text{Spectator factor} = \frac{1 + 0.02}{2} = 0.510, \quad (6.4)$$

$$W = 0.871 \times 0.510 = 0.444. \quad (6.5)$$

The witness dropped from 1.0 to 0.44—a clear signal that quality has degraded significantly.

6.3 Witness vs. Fidelity

Fidelity measures the overlap between the actual state and the ideal state: $F = |\langle T_L | \rho | T_L \rangle|$. It requires full knowledge of the density matrix (state tomography), which is expensive.

The *witness* is a proxy that can be estimated from just three expectation values. It is not identical to fidelity, but it tracks quality faithfully and is operationally efficient.

Chapter 7

Noise and the Hardware Reality

7.1 Sources of Noise

On real quantum hardware, errors arise from:

1. **Gate errors:** Imperfect implementation of unitary operations. Two-qubit gates are the worst offenders, with error rates of 10^{-3} to 10^{-2} .
2. **Readout errors:** Misidentification of $|0\rangle$ vs. $|1\rangle$ during measurement.
3. **Decoherence:** The qubit loses its quantum properties over time (T_1 relaxation and T_2 dephasing).
4. **Cross-talk:** Operations on one qubit inadvertently affect neighbouring qubits.

7.2 Noise Models and Simulators

Since access to real quantum hardware is limited and expensive, we use *noise-model simulators*. The project uses Qiskit Aer's `AerSimulator` with noise models extracted from real IBM backends:

- **fake_brisbane:** A 127-qubit noise model mimicking the IBM Brisbane processor, with realistic gate error rates, readout errors, and T_1/T_2 times.

Intuition

A noise-model simulator is like a flight simulator: it reproduces the conditions of the real thing (including turbulence) without the risk or cost of actual flight time. Results are statistically realistic, even though no quantum hardware is involved.

7.3 Transpilation: From Logical to Physical

The circuit you write in Qiskit uses abstract gates like H , CNOT, and T . But real hardware only supports a limited set of *native gates* (e.g. IBM's basis: $\{ECR, R_Z, S_X, X\}$). The *transpiler* converts your circuit into native gates:

1. **Gate decomposition:** $H \rightarrow S_X \cdot R_Z(\pi/2)$, etc.
2. **Qubit routing:** Map logical qubits to physical qubits on the hardware's connectivity graph. Insert SWAP gates where needed.
3. **Optimisation:** Cancel redundant gates, merge rotations, simplify sequences.

Qiskit provides optimisation levels 0–3:

Level	Description
0	No optimisation (just decomposition and routing)
1	Light optimisation (default)
2	Medium: gate cancellation, commutation analysis
3	Heavy: resynthesis of 2-qubit blocks

Higher Is Not Always Better

Aggressive optimisation reduces gate count but may reroute qubits onto noisier connections. The *net* effect depends on the specific circuit and the specific hardware topology. The ratchet explores multiple optimisation levels to find the empirically best choice.

7.4 Cost Model

The *cost* of a circuit quantifies its resource consumption. The project uses a weighted sum:

$$\text{cost} = w_{2q} \cdot n_{2q} + w_{\text{depth}} \cdot d + c_0, \quad (7.1)$$

where n_{2q} is the two-qubit gate count, d is the circuit depth, and c_0 is a baseline cost. The weights come from the rung configuration and can be tuned.

Two-qubit gates dominate: they are the noisiest operations, so w_{2q} is typically the largest weight.

Chapter 8

Scoring: Putting It All Together

An experiment produces several metrics: witness value, acceptance rate, circuit cost. We need a single number to compare experiments. The *score* does this.

8.1 The Weighted Acceptance-Cost Score

For rungs 1–3, the score is:

$$\text{score} = \frac{\text{quality} \times \text{acceptance_rate}}{\text{cost}} \quad (8.1)$$

where *quality* is the magic-state witness W .

Intuition

The score answers: “How much magic-state quality do I get per unit of resource spent, accounting for the shots I had to throw away?” It naturally balances three competing concerns:

- **Quality:** higher witness is better.
- **Acceptance:** fewer discarded shots is better.
- **Cost:** simpler circuits are better (cheaper to run).

Score Tension

A circuit that is $2\times$ better in quality but $3\times$ more expensive scores *worse*:

$$\frac{2q \cdot a}{3c} = \frac{2}{3} \cdot \frac{qa}{c} < \frac{qa}{c}.$$

The score penalises complexity unless it delivers proportionally more quality.

8.2 Factory Throughput Score

For rungs 4–5, the score shifts to a *factory throughput* model: how many usable T -states per unit time can the circuit produce?

$$\text{factory_score} = \text{quality} \times \text{acceptance_rate} \times \frac{1}{\text{cost}^{3/2}}. \quad (8.2)$$

The steeper cost penalty ($\text{cost}^{3/2}$ instead of cost) reflects the operational reality that in a T -state factory, cost compounds: each round of distillation consumes multiple copies.

8.3 Failure Modes

Three things can go wrong, in increasing order of severity:

1. **High cost:** The circuit is expensive but still works. Fix: optimise transpilation settings.
2. **Poor acceptance:** Many shots are rejected. This wastes compute but the accepted shots may still be good. Fix: reduce noise exposure (fewer gates, better layout).
3. **Low magic witness:** The T -state character itself is lost. Even the accepted shots produce poor quality. This is the most severe failure—the experiment has failed its fundamental purpose.

Chapter 9

The Ratchet: Learning by Doing

The ratchet is an automated optimisation system inspired by Andrej Karpathy’s “autoresearch” philosophy: let the system run experiments, learn from the results, and improve its own configuration.

9.1 The Incumbent-Challenger Model

The Ratchet Guarantee

The ratchet maintains an *incumbent*—the best configuration found so far. In each step, it generates *challengers* (alternative configurations) and evaluates them. A challenger replaces the incumbent *only if it scores strictly higher*. The incumbent never gets worse.

This monotonicity guarantee is the defining property of a ratchet (named after the mechanical device that turns in only one direction). It means the search is safe: you can stop at any time and your best result is preserved.

9.1.1 The Bootstrap Incumbent

The first incumbent is not random—it is a *bootstrap incumbent*: a hand-picked, domain-expert guess at reasonable default parameters. This warm start means the ratchet begins from a sensible baseline rather than wasting time on obviously bad configurations.

9.2 Challenger Generation Strategies

The ratchet generates challengers using three strategies, each with a budget allocation:

9.2.1 NeighborWalk (40% of budget)

Changes *exactly one parameter* at a time, trying every alternative value for that parameter while keeping all others fixed.

- **Strengths:** Systematic, guaranteed to find all single-parameter improvements.

- **Weakness:** Blind to *parameter interactions*—it cannot discover that changing two parameters simultaneously produces a synergy that neither change alone would find.

9.2.2 RandomCombo (30% of budget)

Mutates *multiple parameters simultaneously* at random.

- **Strengths:** Can discover multi-parameter interactions and escape local optima.
- **Weakness:** Less systematic—relies on luck to find good combinations.

9.2.3 LessonGuided (30% of budget)

Uses rules extracted from previous experiments to focus the search.

- **Strengths:** Exploits accumulated knowledge, avoids repeating known mistakes.
- **Weakness:** Only available after lessons have been extracted (not on the first rung).

Intuition

Think of the three strategies as three employees searching a warehouse: NeighborWalk checks one shelf at a time (thorough but slow). RandomCombo wanders around trying random combinations (creative but unpredictable). LessonGuided reads the notes from previous searches first (efficient but needs prior experience).

9.3 Ratchet Steps and Rungs

A **step** is one round of challenger generation and evaluation. In each step:

1. Generate a batch of challengers using the three strategies.
2. Evaluate each challenger (run the experiment, compute the score).
3. If any challenger beats the incumbent, replace the incumbent.
4. Log the result (winner, margin, all scores).

A **rung** is a sequence of steps, terminated when the *patience* is exhausted:

Definition 9.1 (Patience). If p consecutive steps fail to improve the incumbent, the rung stops. This prevents wasting compute once the nearby parameter space has been exhausted.

9.4 Lesson Extraction

After each rung, the system analyses all experimental results and extracts *lessons*—rules about which parameter values help or hurt:

1. **Fix rules:** “Always use this value”—a parameter value that consistently appears in top-scoring experiments.

2. **Avoid rules:** “Never use this value”—a parameter value that consistently appears in bottom-scoring experiments.

Lessons are stored in two formats:

- A human-readable *narrative* (natural language summary).
- Machine-readable *SearchRules* (JSON) that the LessonGuided strategy can consume directly.

9.5 Search Space Narrowing

Lessons also *narrow* the search space for subsequent rungs. If a value is consistently bad, it is removed from the allowed options. The dimension remains (the parameter still exists), but with fewer values to explore. A minimum number of values per dimension is preserved to prevent overfitting to noise.

9.6 Cross-Rung Propagation

The winning configuration from rung N is *propagated* as the bootstrap incumbent for rung $N + 1$. This avoids cold-starting each rung and allows the system to build on previous gains.

Combined with lesson extraction and space narrowing, this creates a *progressive refinement* loop:

$$\text{Run} \rightarrow \text{Learn} \rightarrow \text{Narrow} \rightarrow \text{Propagate} \rightarrow \text{Run} \rightarrow \dots$$

9.7 Transfer Evaluation

A configuration optimised for one backend might be overfitted to that backend’s specific noise profile. *Transfer evaluation* tests the winning configuration on a different backend:

- If the score is similar, the configuration is *robust*.
- If the score drops sharply, the configuration is *overfitted* to the source backend.

Transfer evaluation is a form of generalisation testing, analogous to validating a machine-learning model on a held-out dataset.

Chapter 10

The Five Rungs

The system organises its search into five *rungs*, each adding complexity:

Rung	Focus	Backend	Scorer
1	Core parameters (seed, encoder, verification)	fake_brisbane	WAC
2	Transpilation (opt level, layout, routing)	fake_brisbane	WAC
3	Fine-tuning (approximation degree, initial layout)	fake_brisbane	WAC
4	Hardware-aware optimisation	fake_brisbane	Factory
5	Transfer validation	Different backend	Factory

Each rung inherits the best configuration from the previous one, narrows the search space based on lessons, and explores the next layer of parameters. The first three rungs use the Weighted Acceptance-Cost (WAC) score; rungs 4–5 switch to factory throughput to reflect production-oriented priorities.

Chapter 11

Putting It All Together: The Pipeline

Here is the complete flow from start to finish:

1. **Configuration:** Load a rung YAML file specifying the parameter space, scorer, backend, and budget.
2. **Preparation:** Build the quantum circuit from the experiment spec (seed style \rightarrow encoder \rightarrow verification circuits \rightarrow witness circuits).
3. **Transpilation:** Compile the circuit for the target backend at the specified optimisation level.
4. **Execution:** Run the circuit for N shots on the noise-model simulator.
5. **Analysis:**
 - (a) Parse syndrome bits to compute acceptance rate.
 - (b) Parse data bits on accepted shots to compute $\langle X_L \rangle$, $\langle Y_L \rangle$, $\langle Z_{\text{spec}} \rangle$.
 - (c) Compute the witness W .
 - (d) Compute the cost from circuit metrics.
 - (e) Compute the score.
6. **Ratchet:** Compare challenger scores to the incumbent. Promote the winner. Extract lessons. Narrow the search space. Propagate to the next rung.
7. **Transfer:** Test the final configuration on a different backend.

Where to See This in the Notebooks

- **Plan A, Notebook 01:** Steps 1–5 in detail (one cell per stage).
- **Plan A, Notebook 02:** Step 5 in depth (scoring, parameter sweeps).
- **Plan A, Notebook 03:** Steps 6–7 (the ratchet in action).
- **Plan B (Spiral):** All steps in three passes of increasing depth.
- **Plan C, Track A:** Steps 1–3 (physics focus).
- **Plan C, Track B:** Steps 3–5 (engineering focus).
- **Plan C, Track C:** Steps 6–7 (optimisation focus).
- **Plan C, Dashboard:** Interactive exploration of step 2 parameters.
- **Plan D, Experiment 1:** Steps 1–3 (encoding and error detection, ideal simulator).
- **Plan D, Experiment 2:** Steps 3–5 (noise, scoring, parameter sweep).
- **Plan D, Experiment 3:** Steps 6–7 (ratchet, lessons, transfer evaluation).

Chapter 12

Glossary

Acceptance rate

Fraction of shots that pass the syndrome check (postselection).

Ancilla

An auxiliary qubit used for syndrome extraction without disturbing the data qubits.

Bloch sphere

A geometric representation of a single-qubit state as a point on a unit sphere.

Bootstrap incumbent

The hand-picked initial configuration that the ratchet starts from.

Challenger

A candidate configuration that competes against the incumbent in a ratchet step.

Clifford group

The group of gates generated by $\{H, S, \text{CNOT}\}$. Classically simulable by the Gottesman–Knill theorem.

Codespace

The subspace of the physical Hilbert space where valid codewords live. Defined by the simultaneous $+1$ eigenspace of all stabilisers.

Cost

A scalar measuring the resource consumption of a circuit (dominated by two-qubit gate count).

Distance

The minimum weight of an undetectable error. For $[[4, 2, 2]]$, $d = 2$: all weight-1 errors are detectable.

Eastin–Knill theorem

No quantum code admits a universal set of transversal gates.

Factory throughput

A scoring function that penalises cost more heavily, modelling a T -state production pipeline.

Fidelity

$F = |\langle \psi | \rho | \psi \rangle|$: the overlap between the actual state and the ideal target.

Global phase

A factor $e^{i\gamma}$ multiplying the entire state vector. Unphysical and unmeasurable.

Gottesman–Knill theorem

Clifford-only circuits can be efficiently simulated on a classical computer.

Incumbent The best configuration found so far. Replaced only when a challenger scores strictly higher.

Lesson A rule extracted from experimental results (“fix” or “avoid” a parameter value).

Logical operator

An operator that acts on the encoded (logical) information within the codespace.

Magic state The state $|T\rangle = (|0\rangle + e^{i\pi/4}|1\rangle)/\sqrt{2}$, consumed to implement the T gate via gate teleportation.

Narrowing Removing poorly-performing parameter values from the search space between rungs.

No-cloning theorem

No unitary operation can copy an unknown quantum state.

Patience The number of consecutive no-improvement steps before a rung terminates.

Postselection Discarding shots where the syndrome indicates an error.

Ratchet A monotonic optimiser: the incumbent never gets worse.

Rung A stage of the optimisation pipeline with specific parameters to explore and a fixed budget.

Seed style The gate sequence used to prepare the magic state on a single qubit before encoding.

Shot One execution of the full circuit (preparation + measurement).

Stabiliser A Pauli operator whose +1 eigenspace defines the codespace. For $[[4, 2, 2]]$: $XXXX$ and $ZZZZ$.

Syndrome The measurement outcome of the ancilla qubits, indicating whether an error has been detected.

Transpilation Converting a logical circuit into native gates for a specific hardware backend.

Transfer evaluation

Testing a configuration on a different backend to check for overfitting.

Witness A scalar quantity computed from expectation values that estimates the quality of the prepared magic state.

Appendix A

Mathematical Background

This appendix collects the mathematical prerequisites. Skip it if you are comfortable with linear algebra over \mathbb{C} .

A.1 Complex Numbers and Amplitudes

A complex number $z = a + bi$ has a real part a , imaginary part b , magnitude $|z| = \sqrt{a^2 + b^2}$, and phase $\arg(z) = \arctan(b/a)$. In polar form: $z = |z|e^{i\theta}$.

Quantum amplitudes are complex numbers. The probability of measuring a state is the squared magnitude of its amplitude: $p = |\alpha|^2$.

A.2 Tensor Products

The state space of n qubits is the tensor product $(\mathbb{C}^2)^{\otimes n}$, with dimension 2^n . For two qubits:

$$|\psi\rangle \otimes |\phi\rangle = \begin{pmatrix} \alpha_\psi \\ \beta_\psi \end{pmatrix} \otimes \begin{pmatrix} \alpha_\phi \\ \beta_\phi \end{pmatrix} = \begin{pmatrix} \alpha_\psi \alpha_\phi \\ \alpha_\psi \beta_\phi \\ \beta_\psi \alpha_\phi \\ \beta_\psi \beta_\phi \end{pmatrix}. \quad (\text{A.1})$$

A.3 Pauli Matrices

The four Pauli matrices form a basis for 2×2 Hermitian matrices:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \quad (\text{A.2})$$

Key properties:

- $X^2 = Y^2 = Z^2 = I$.
- $XY = iZ, YZ = iX, ZX = iY$ (cyclic).

- $XZ = -ZX$ (anti-commutation). This is why $ZZZZ$ detects X errors.

A Pauli string like $XXXX$ is the tensor product $X \otimes X \otimes X \otimes X$, a 16×16 matrix acting on 4 qubits.

A.4 Eigenvalues and Expectation Values

The *expectation value* of an observable O in state $|\psi\rangle$ is:

$$\langle O \rangle = \langle \psi | O | \psi \rangle. \tag{A.3}$$

For a Pauli operator P with $P^2 = I$, the eigenvalues are exactly ± 1 . A measurement of P always returns $+1$ or -1 . The expectation value is the average over many measurements.

Appendix B

Notebook–Compendium Cross-Reference

Notebook Topic	Notebooks	Compendium
T-state definition & Bloch sphere	A/01 §1–2, B §2.1, C/A §1–3	chapter 3
Why encode (no-cloning, distance)	A/01 §3, C/A §1	chapter 4 §1–2
Stabilisers & codespace	A/01 §6, B §2.3, C/A §4	chapter 4 §3
Logical operators	A/01 §6, C/A §5	section 4.4
Encoder circuits	A/01 §4–5, C/A §6	section 4.5
Error detection	A/01 §7, C/A §8	section 4.6
Ancilla & syndrome extraction	A/01 §9, C/A §7	chapter 5 §2
Postselection	A/01 §11, A/02 §3, B §2.5	section 5.3
Noise models & transpilation	A/02 §2, C/B §1–3	chapter 7
Magic witness formula	A/02 §5, B §2.7, C/A §9	chapter 6
Scoring formula	A/02 §7, B §2.9, C/B §8	chapter 8
Factory throughput	A/02 §10, C/B §9	chapter 8 §2
Failure modes	A/02 §9, C/B §7	section 8.3
Ratchet mechanism	A/03 §1–4, B §2.10–12, C/C §1–7	chapter 9 §1–3
Search strategies	A/03 §7, B §3.5, C/C §3–4	section 9.2
Lesson extraction & rules	A/03 §8, B §3.6, C/C §8–9	section 9.4

End of compendium.