END QUBIT METHOD

Radian-Transition Spectator Feedback for Predictable Qubit Phase Coherence

Company technical paper, reproducibility specification, IBM hardware screen and patent-counsel work package

EVIDENCE **STATUS**

Frozen simulations passed their stated model checks. No IBM hardware result is claimed in this document. Hardware language is conditional on the included notebook returning a preregistered PASS.

Jordan Ryan Evans

Independent research program | Medicine Hat, Alberta, Canada

Version 1.0-draft | 6 August 2026

Document control and legal-use boundary

NO HARDWARE ASSUMPTION MAY BE REPORTED AS FACT

The paper includes exact wording for a future pass because the requested IBM time is short. The values in that wording must come from END_QM_hardware_decision.json. Simulated values may never fill hardware fields.

Item	Value
Document	END Qubit Method long-form technical paper
Author / proposed inventor	Jordan Ryan Evans
Evidence freeze	Simulation seed 20260806; hardware prospective
Primary implementation	END-QM-6, public IBM dynamic circuit
Secondary implementation	Telemetry lexicon; lab-only END-QM/Dressed
License posture	Internal confidential evaluation only; signed terms recommended
Patent status	Unresolved; do not state patent pending until filed

A paper cannot guarantee ownership or acceptance. Copyright protects the text/code expression, not the underlying idea. Patent rights depend on filing, inventorship, disclosure, enablement, novelty and non-obviousness. Trade-secret rights depend on reasonable secrecy. Contract rights depend on agreed terms. Counsel should review this package before public or company disclosure.

Abstract

We disclose END-QM, a confidence-gated phase-feedback method that converts two time-ordered, multi-axis sentinel-qubit observations into a bounded virtual-Z update on a separate data qubit. Three sentinels implement a six-sector radian encoder at two boundaries. The two three-bit observations are packed as a transition code; a pilot-only circular lexicon maps supported codes to conditional data phase. Ambiguous/low-support codes fall back to zero, and a reserved pilot-validation subset disables any local map that does not improve estimated coherence. A held-out comparison uses a latency-matched open-loop circuit, wrong-sign, raw-idle and XY4 controls across 12 start-phase windows. In a frozen synthetic common-mode drift model, END-QM achieved a 1.2219× median normalized equatorial-coherence AUC gain over matched open loop; the weakest window was 1.1055× and its simultaneous lower bound 1.0968×. These values are model-specific and not hardware evidence. A 96-circuit pilot and randomized 480-circuit confirmation notebook targets current IBM Runtime dynamic circuits within three requested job caps totaling 540 seconds. The package defines falsification, replication and IP-review boundaries and preserves an earlier richer-telemetry forecast separately.

Keywords: quantum control; dephasing; spectator qubits; dynamic circuits; virtual-Z; circular statistics; phase feedback; radian lexicon; reproducibility.

Contents

Contents — continued

Notation and abbreviations

Symbol / term	Meaning
END-QM	END Qubit Method family
END-QM-6	Six-sector two-frame public-hardware embodiment
QDS / ΔQDS	Qubit dressed-state modulation / detuning-modulated variant
φ, Δφ	Wrapped phase and phase transition in radians
f, Δf	Frequency and frame-to-frame frequency change
c	Six-bit raw transition code
u_c	Bounded correction for code c
R	Circular resultant/concentration
AUC	Area under normalized coherence curve
matched_open	Same sensing/reset/latency as END-QM, with no correction
oracle	Simulation-only use of inaccessible true common phase
MCM	Mid-circuit measurement
VZ	Virtual-Z phase-frame update

1. Executive decision brief

END-QM is presented as a reproducible research asset, not as a guaranteed cure for decoherence. Its narrow claim is that a data qubit can sometimes retain more equatorial coherence when a repeatable, common-mode phase component is observed on separate sentinel qubits and converted into a bounded, independently validated frame correction. The method cannot restore energy lost through T1 relaxation, and it is expected to fail when the dominant noise is local, uncorrelated, too fast for the measurement latency, or too weakly observed.

The company-facing package separates three evidence levels. First, the full telemetry simulation inherited from QDC-RAFS shows what a high-quality external phase estimate could enable under a deliberately slow common-detuning model. Second, END-QM-6 models the public IBM embodiment with binary sentinels, ambiguous states, finite support, wrong-sign and shuffled controls, and a pilot-validation gate. Third, the IBM notebook executes the causal hardware screen. Only the third can support a hardware statement, and even a pass is limited to one device and calibration interval until replicated.

The requested commercial posture is therefore staged. A recipient can audit the mathematics and code without accepting a valuation; execute a short controlled screen; reproduce on additional dates/qubits; then negotiate an evaluation option or license if the result persists. This ordering reduces the chance that an enthusiastic simulation statement is mistaken for hardware evidence or a binding guarantee. It also creates a better record for patent counsel because the claimed combination, implementation details, failure modes and known prior-art pressure are stated before selective hardware reporting.

2. Claim taxonomy and non-claims

The phrase “extends a qubit state” is ambiguous. A method may rotate the observed phase, suppress an ensemble dephasing envelope, improve a task-specific survival probability, reduce coherent error, or change an exponential fit without altering intrinsic materials limits. END-QM therefore uses equatorial coherence and a fixed-grid normalized AUC as its primary observable. It does not rename that metric T2. A T2-style fit may be reported secondarily only with the exact fit window, model, weighting and uncertainty.

A virtual-Z operation is an efficient phase-frame update [1]. By itself it cannot increase the radial length of a Bloch vector; a precomputed virtual-Z schedule can merely rotate measurement phase. END-QM requires information that is correlated with the data-qubit phase error and a correction selected causally from that information. This is why the public embodiment measures separate sentinels and why the latency-matched open-loop circuit is the primary control. If END and matched open loop are indistinguishable, the method has not shown a feedback benefit.

The package makes no promise that IBM will accept a paper, buy a method, provide more time, or endorse the result. Scientific merit is determined by the protocol, data and replication; business interest is a separate decision. It also makes no claim that SMART was copied from Evans's work. The accessible SMART publication predates the supplied 2026 records [3]. Any earlier private provenance would need independently dated evidence and legal review.

- Claimable after simulation only: code executed as specified; frozen model behavior; unit-test results.
- Claimable after a one-device pass: only the named device/date/qubits and preregistered metric.
- Not claimable without replication: general coherence extension, scalability, fault-tolerance benefit, valuation or vendor acceptance.
- Never claim: reversal of T1 loss, universal decoherence solution, theft, patentability or freedom to operate without evidence.

3. Conceptual provenance from Evans's END, radian lexicon and QDS work

Evans's supplied materials repeatedly emphasize phase anchors, frame-to-frame changes, frequency and change in frequency, circular rather than linear phase handling, and an operational lexicon that maps recurrent patterns to corrections. The useful technical kernel is not a broad cosmological interpretation; it is the recognition that a periodic state must be estimated on a circle, that successive frames contain directional information, and that an observed change should actuate a later frame rather than be retroactively applied.

The QDS and ΔQDS materials add a control-system language based on a rotating-frame Hamiltonian with detuning, drive amplitude and drive phase. Their earlier Markovian simulation reported approximately 54.55 microseconds for a baseline fit and 78.63 microseconds for a ΔQDS fit, but the inferred ratio changed substantially with the fit window: a narrow window could yield about 2.12 while a broader 10–120 microsecond window yielded about 1.01. That sensitivity is evidence for stricter metrics, not evidence for a universal factor. END-QM keeps the Hamiltonian as a laboratory embodiment and replaces fit-driven screening with fixed-grid AUC.

The newer END monograph's explicit withdrawal of unsupported empirical claims is adopted as a methodological constraint. Frame count is not physical time unless calibrated; radian labels are coordinates, not mechanisms; pattern matching in unrelated time series does not prove quantum control. The present paper uses the lexicon only where a physical sensor, a causal data path and a falsifiable control exist.

4. Decoherence problem decomposition

For a data qubit, relaxation and dephasing must be separated. T1 processes change population and bound recoverable transverse coherence. Dephasing includes deterministic offsets, slowly wandering frequencies, common-mode fields, local stochastic noise, drive-induced shifts and measurement back-action. A feedback method is valuable only for the component that is both observable soon enough and correlated with the data error. Unobserved local noise remains in the residual channel.

Dynamical decoupling reshapes sensitivity to environmental spectra [2]. It can be highly effective without measuring the environment, but finite pulses, calibration, idle scheduling and hardware-specific noise affect its result [7]. Real-time Hamiltonian estimation and predictive compensation demonstrate that measured correlations can suppress dephasing in suitable systems [4,5]. Spectator-qubit methods likewise establish that separate quantum sensors can reveal shared disturbances [6]. END-QM is positioned within this known control landscape, not outside it.

The design challenge is therefore selective correction. If the controller reacts to low-support or ambiguous sensor outcomes, estimation noise becomes applied phase noise. If the controller learns and evaluates on the same shots, apparent gains can be optimistic. If it is compared only with raw idle, measurement latency can be misread as a feedback effect. END-QM addresses these issues with ambiguity fallback, support gates, a reserved pilot-validation subset and a matched dynamic-circuit control.

5. Rotating-frame model

In the rotating frame and under the usual two-level approximation, a driven qubit may be represented by a Hamiltonian containing detuning along z and a transverse drive in the equatorial plane. The END-QM public embodiment uses only calibrated gates, delays, measurement, reset and virtual-Z conditionals. The laboratory embodiment can expose the continuous controls explicitly.

Let the data phase accumulated during storage be the sum of a common component, an independent component and deterministic start phase. The sentinels measure a noisy function of the common component at two boundaries. The controller receives no privileged access to the true phase. The oracle curve in simulation uses true common phase only to show an unattainable upper bound; it is never part of the implementable result.

A control update changes the data-frame phase but cannot undo amplitude damping. With T1 envelope A(t), the observed equatorial vector is A(t) times the circular mean of the residual phase. This decomposition makes the limitation visible: even perfect common-phase cancellation approaches the oracle envelope but cannot exceed the T1-limited amplitude represented in the model.

6. Radian state, frequency change and causal timing

The physical premise is that the phase advance between two timestamps depends on the integral of instantaneous detuning. When frequency changes, the next phase cannot be inferred from a single instantaneous phase alone. END-QM retains an ordered pair of observations. In the telemetry form this state may include measured frequency f_k and difference Δf_k. In END-QM-6 the ordered binary patterns act as a coarse proxy for the same transition.

Causality imposes an important boundary. A sensor observation at frame k may affect only operations scheduled after the corresponding classical result is available. The IBM circuit therefore keeps the data qubit alive across two sentinel measurements, performs the conditional virtual-Z update after the second measurement, and then leaves a final storage segment before analysis. The matched control preserves the same mid-circuit operations but omits the correction.

The two boundaries are fixed at 0.34 and 0.70 of the requested storage duration. They are not asserted to be globally optimal. They create a measurable transition and leave 30% of the nominal interval after actuation. A future design can optimize boundaries prospectively, but changing them after confirmation results would create a new protocol version.

7. Three-sentinel six-sector encoder

A single binary Ramsey observation supplies too little angular information for a one-shot phase label. END-QM-6 prepares three sentinel qubits with analysis axes separated by 120 degrees. Across the circle, six non-unanimous three-bit patterns dominate ideal sector responses. The unanimous patterns 000 and 111 are treated as ambiguous rather than forced into a sector.

This encoder trades precision for compatibility. It uses no two-qubit gate and therefore does not require connectivity between the data qubit and sentinels. It does require common-mode sensitivity: physical separation, frequency mismatch and local noise can reduce correlation. The notebook records selected qubits and calibration but cannot guarantee spatially common noise. A hardware failure may therefore indicate absent observability rather than an incorrect software implementation.

Two three-bit observations are packed into a raw code $c=s_0+8s_1$. Packing preserves both direction and endpoint without assuming arithmetic inside the dynamic circuit. IBM Runtime hardware supports classical conditions but places restrictions on arithmetic and nested flow [10]. The notebook therefore compiles a finite set of direct equality conditions, each applying a previously frozen angle.

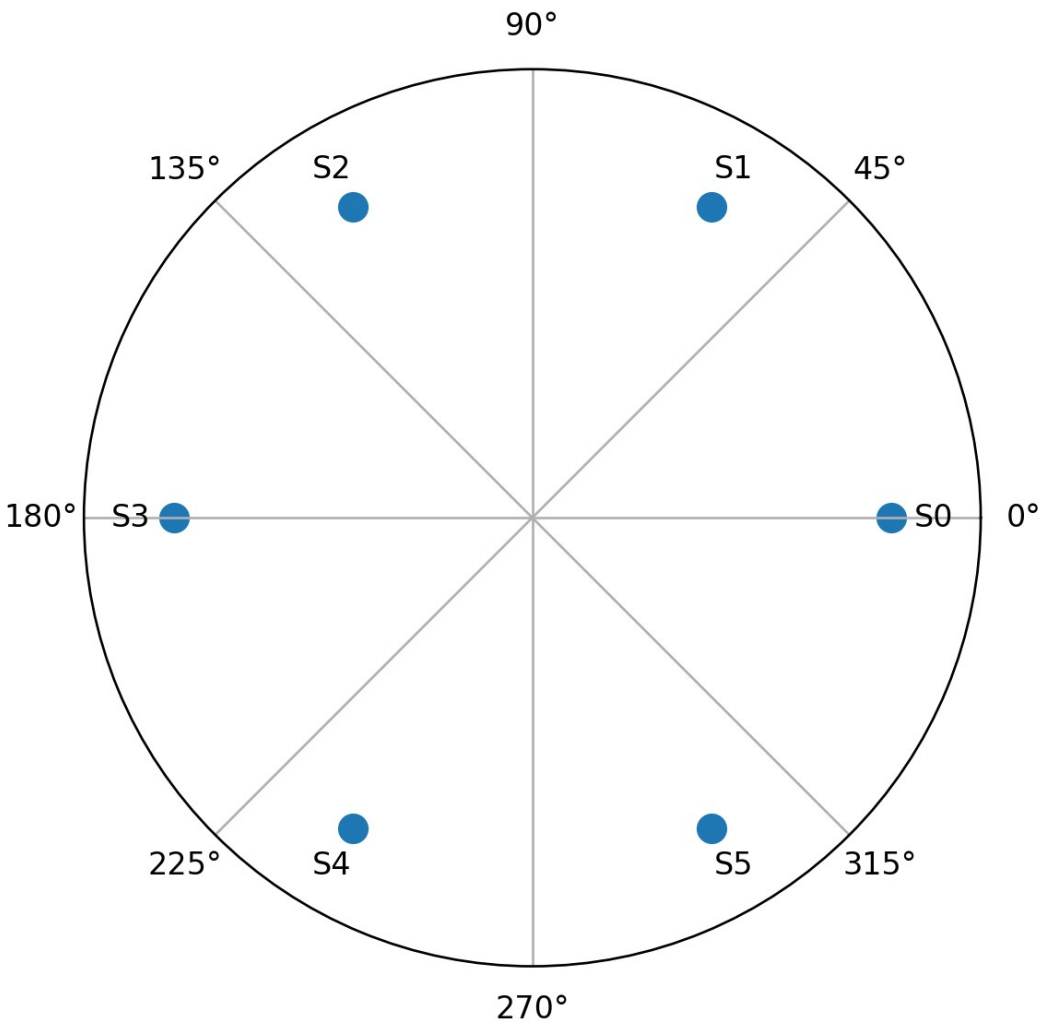

Figure 1. Six-sector encoding; unanimous patterns are fallback states.

8. Circular lexicon and confidence structure

For each start window, duration and transition code, pilot X/Y tomography estimates a conditional complex vector m_x+i m_y. Its argument is the predicted data phase, while its magnitude summarizes conditional concentration and readout attenuation. The proposed correction is the negative angle, clipped to ±π/2. A code with insufficient X or Y support, an undefined vector, or an ambiguous half-state receives zero.

The local support rule prevents a rare transition from carrying a large phase based on a handful of outcomes. It does not prove that the map generalizes. END-QM therefore reserves 40% of pilot shots. The entire window-duration map is evaluated on those untouched pilot-validation shots; if it does not beat no correction, every angle in that local map is set to zero. This fail-closed gate is applied before confirmation circuit compilation.

The hardware notebook records both pre-gate and final corrections. An evaluator can therefore distinguish a physically unobserved sector, a low-support sector and an apparently harmful full map. The fallback rate is a result, not an inconvenience. A method that reports gains only after discarding fallback-heavy windows would violate the all-window design.

9. Controller and circuit semantics

The dynamic circuit begins with an equatorial data state and three equatorial sentinels under the selected start-phase window. After the first delay segment, each sentinel is rotated into its analysis axis and measured. The sentinels are reset and re-prepared, then measured again after the second segment. The six classical bits select at most one bounded virtual-Z update on the data qubit. The final delay and X/Y analysis follow.

Mid-circuit measurement is not free. It may introduce hundreds of nanoseconds or more of latency, plus dephasing, reset error and crosstalk. For this reason, `matched_open` executes the same sensing and reset sequence with no conditional update. `wrong_sign` applies the frozen angle in the opposite direction. A genuine phase-cancellation mechanism should outperform both. Raw idle and XY4 help contextualize the engineering cost but do not replace the matched causal contrast.

The public notebook uses Qiskit 2.5-era circuit control flow and qiskit-ibm-runtime SamplerV2 conventions documented by IBM [9,10]. It does not call Qiskit Pulse because IBM removed public pulse-level control and Qiskit 2.0 removed the pulse module [15]. This boundary avoids presenting code that cannot run on the intended service.

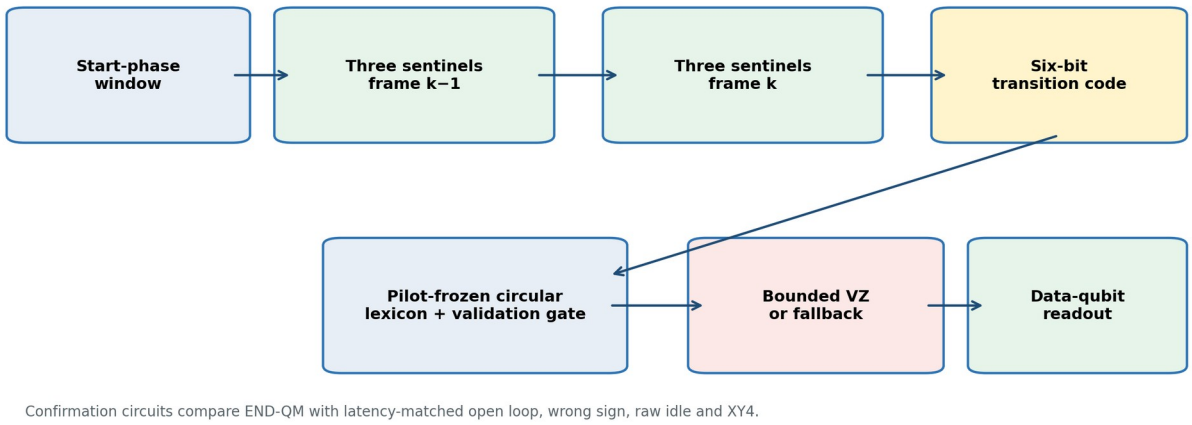

Figure 2. END-QM-6 causal architecture and matched confirmation structure.

10. Dressed-state extension

The lab-only END-QM/Dressed embodiment combines the conditional radian update with ΔQDS-style modulation of detuning, amplitude and drive phase. The purpose is not to rename SMART or claim continuous driving broadly. It is to test whether a slowly varying dressed basis can reduce sensitivity while the discrete lexicon cancels a residual predictable component. These mechanisms may interact constructively or destructively.

A proper laboratory test must calibrate Rabi rate, detuning amplitude, modulation frequency, phase relation, leakage outside the computational subspace, AC Stark shift, power heating and pulse distortion. It must include continuous-drive-only, feedback-only, combined, wrong-sign and equal-power controls. A combined improvement cannot be assigned to the lexicon unless the feedback-only and wrong-sign contrasts are retained.

The supplied QDS simulation's strong fit-window dependence is a warning. The dressed protocol should use a preregistered time grid and AUC, then optionally fit physical models with residual diagnostics. Any claim that a modulation “extends T2” must report whether the estimate is Ramsey T2*, echo T2, driven coherence, or task-specific survival.

11. Synthetic model design

The END-QM-6 simulation is a mechanism forecast with common static detuning, linear slope, a synchronous sinusoidal component, independent data detuning, phase diffusion, finite sentinel noise and T1 attenuation. It does not simulate a named IBM backend. Parameters were frozen before the final held-out run; 30,000 pilot trajectories and 70,000 independent test trajectories were used for each window-duration cell.

The model gives the controller exactly the type of structure it is designed to exploit: a slow common component measured with high correlation. This makes a positive result plausible by construction, so the model cannot estimate the probability of success on an arbitrary QPU. Its legitimate role is to verify signs, circular wrapping, transition packing, fallback behavior, negative-control ordering, bootstrapping and output generation.

Twelve start-phase windows and five times from zero to 220 microseconds are evaluated. The fixed seed is 20260806. The simulator produces method coherence components, paired covariance, AUC, per-window ratios, simultaneous intervals, robustness sweeps, lexicon entries and an audit sample. Raw held-out phases are retained in compressed form.

12. Frozen END-QM-6 results

The final gated run produced a median normalized coherence-AUC gain of 1.2219× over matched open loop. Its bootstrap 95% interval was 1.2178–1.2248×. The weakest of 12 start-phase windows was 1.1055× and the weakest familywise lower bound was 1.0968×. Every window exceeded parity in this synthetic model.

The median fallback-code fraction was 0.6094. That high fraction is intentional: unanimous sensor halves, rare transitions and pilot maps that fail reserved validation are suppressed. The median latent correlation between common data phase and the second sentinel latent phase was 0.9923, which explains why the forecast is favorable and also shows why it cannot be generalized to hardware without measuring observability.

The median AUC values were 96.23 microseconds for END-QM, 78.76 for matched open loop, 75.12 for wrong sign, 75.09 for the shuffled map and 157.01 for the non-implementable common-phase oracle. The gap to the oracle indicates uncorrected independent noise, T1 loss, coarse sectorization and fallback. The control ordering is consistent with causal correction inside this model; it is not evidence about a QPU.

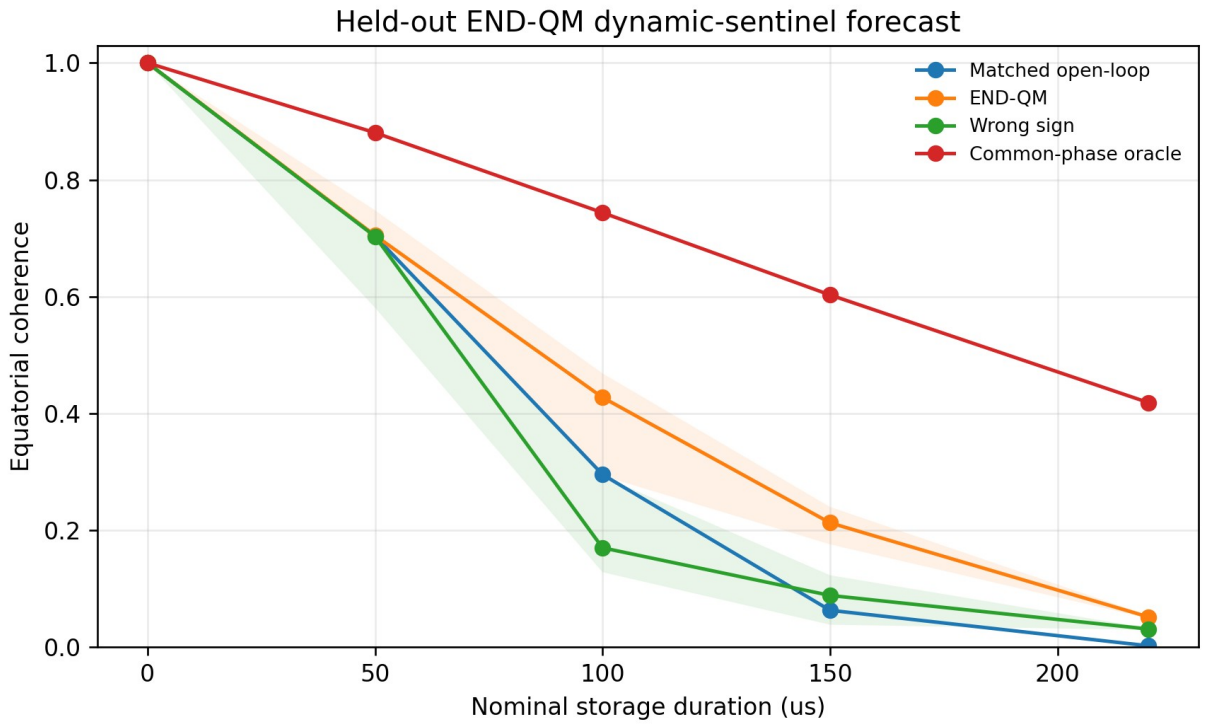

Figure 3. Frozen synthetic END-QM-6 coherence curves. Model result only.

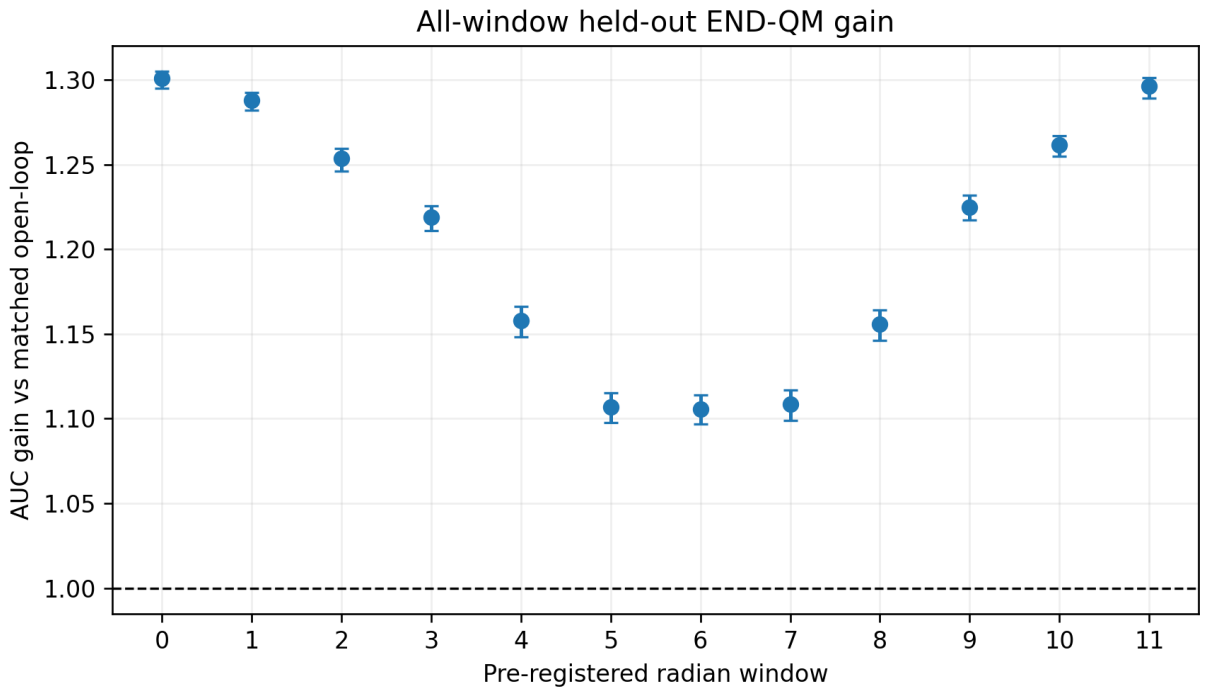

Figure 4. All-window END/matched AUC gains with simultaneous intervals.

13. Earlier telemetry-layer evidence

The preserved QDC-RAFS v0.1 simulation used a richer telemetry estimate rather than binary dynamic-circuit sentinels. Its held-out median AUC gain over unprotected storage was 2.6717×. The weakest window was 2.6290× and the weakest simultaneous lower bound was 2.5053×. At 160 microseconds the reported coherence was approximately 0.708 versus 0.096 unprotected.

That factor should not be attached to END-QM-6 hardware expectations. The telemetry model assumes a useful external estimate and compares primarily with unprotected storage. Its idealized XY4 AUC was about 224.93 microseconds and its QDC-RAFS AUC about 231.23 microseconds, only roughly 2.8% higher. This is the relevant scale when comparing with established suppression under the same slow-noise model.

The earlier model remains valuable as a software provenance layer and as an upper engineering scenario for systems with high-quality frequency telemetry. It is retained with original checksums and verifier output. No number was retroactively changed to match the new dynamic-sentinel simulation.

14. Negative controls and causal specificity

A wrong-sign correction should worsen or fail to improve coherence if the learned phase direction is meaningful. A shuffled map destroys the association between transition state and correction while preserving the angle distribution. An oracle establishes the best possible cancellation of the modeled common component. These controls answer different questions and should not be collapsed into a single baseline.

In the END-QM-6 forecast, both wrong-sign and shuffled maps fall below matched open loop in median AUC, while the oracle remains far above the implementable method. That ordering is stronger evidence of code correctness than a sole positive ratio. Hardware may not reproduce the ordering; a wrong-sign method that performs equally well or better would undermine the claimed causal direction even if END appeared above raw idle.

The IBM notebook implements wrong sign directly. It does not include shuffled map in the 10-minute confirmation allocation because the five-method, all-window design already contains 480 circuits. A paid or extended run should add a preregistered shuffled-map arm and repeated randomized compilations.

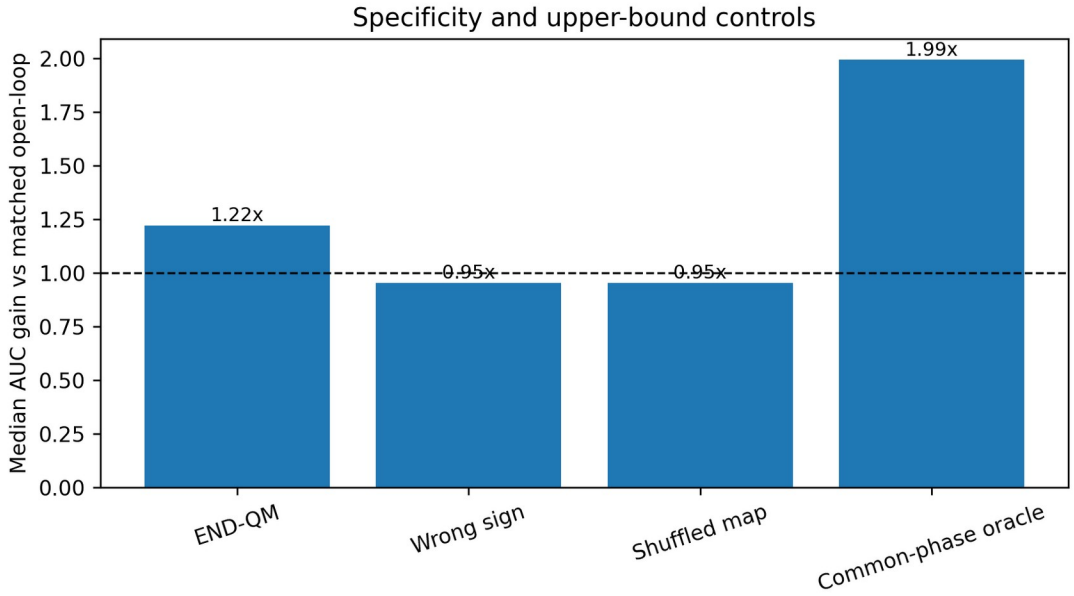

Figure 5. Frozen negative-control ordering and oracle upper bound.

15. Robustness and domain of applicability

The robustness sweep varies common-mode amplitude and sentinel noise. When common structure is absent, the controller should fall back or fail to improve. As common-mode observability grows, the method can cross from parity to benefit. High sensor noise reduces sector reliability and increases the need for support/validation gating. These are domain boundaries, not parameters to tune after hardware results.

The all-window safety gate is conservative but does not make END-QM risk-free. Pilot validation is itself noisy, and repeated opportunities can admit a false-positive map. A production study should use stronger multiplicity control, hierarchical shrinkage, sequential monitoring with error spending, and an independent day-level test. The short screen instead prioritizes transparent falsifiability under a tight QPU budget.

Hardware-specific risks include drift between pilot and confirmation, differing noise at sentinel locations, reset-induced heating, conditional-branch latency, readout bias, compiler changes and phase conventions. Raw circuits, calibration and job IDs are therefore part of the result. A single summary plot is insufficient for technical diligence.

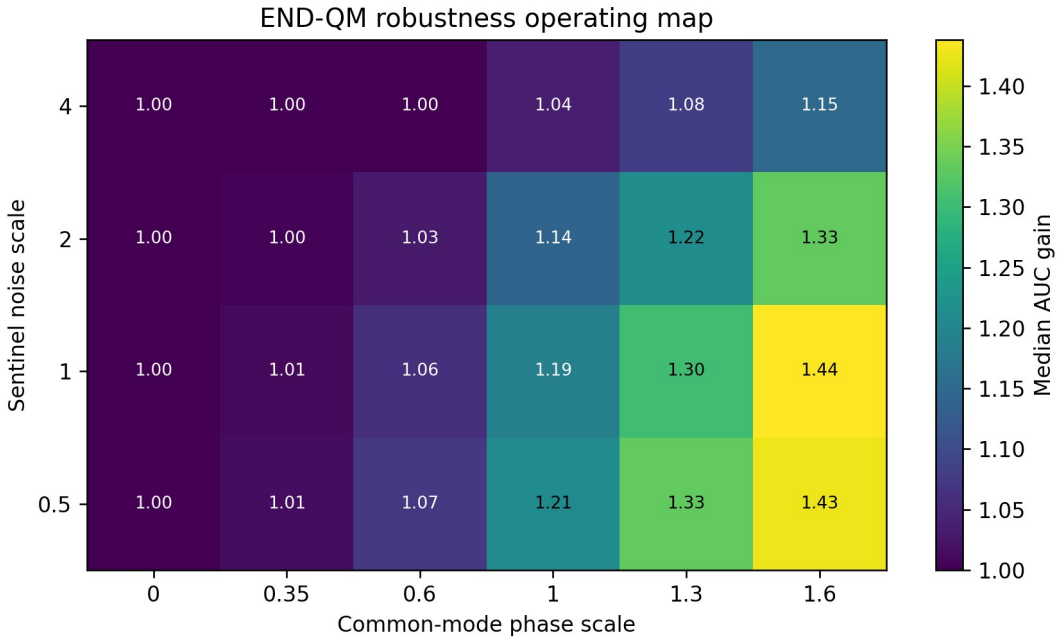

Figure 6. Sensitivity to common-mode amplitude and sentinel noise.

16. Statistical analysis plan

At each method/window/duration, separate X and Y circuits estimate an equatorial vector. The coherence magnitude is normalized by the zero-delay value for the same method and window. AUC is integrated over the adaptively selected but prospectively fixed duration grid. Normalization reduces state-preparation/readout scaling differences, while the matched comparison reduces dynamic-circuit overhead confounding.

The hardware notebook computes a ratio for each of 12 start windows and bootstraps the median by resampling windows. This treats windows as the screening unit; it does not fully model shot-level binomial uncertainty or temporal drift. A subsequent confirmatory analysis should use a hierarchical binomial model or paired bootstrap incorporating shot and circuit layers, report simultaneous intervals for every window, and predefine missing-data behavior.

The threshold of 1.10 is an engineering screening threshold chosen before hardware results. It is not derived from a cost model and does not imply economic materiality. Effect size, interval, control ordering, fallback rate and operational burden should be reported together. P-values without these quantities would not resolve the practical question.

17. IBM hardware experiment

The notebook authenticates with `QiskitRuntimeService(token=key, instance="auto")`, selects an operational QPU, ranks qubits using current T1/T2 and readout error, and chooses four without requiring coupling. Durations are zero plus 0.25, 0.50 and 0.85 of the current data-qubit T2, clipped to 16–220 microseconds. The selected values are printed before submission.

The pilot has 96 circuits at 2,048 shots. Its job requests a 160-second maximum. The 480 held-out circuits are randomized and split into two 240-circuit jobs at 512 shots, each requesting 190 seconds. Requested caps sum to 540 seconds, within a ten-minute target, but the QPU controls actual metering and may reject a plan. IBM documents a rolling Open Plan allowance and job execution limits [11–13].

The code records each job ID immediately before blocking for a result. If Colab disconnects, the job is recovered from the service rather than blindly resubmitted. All raw per-shot sentinel/data records, the pilot split, frozen lexicon, confirmation AUC, decision JSON and plot are downloaded. The executed notebook must be sanitized to remove the API key before sharing.

18. Hardware acceptance and falsification criteria

The screen reports PASS only when the median END/matched AUC ratio is at least 1.10, its paired-window bootstrap lower 95% bound exceeds 1.00, and END exceeds wrong sign. These criteria are conjunctive. A favorable raw-idle comparison cannot rescue failure against the matched control. A favorable subset of windows cannot rescue a failed median or a broad interval.

A PASS permits a narrow sentence naming backend, qubits, calibration date, metric, interval and pending replication. It does not permit the phrases “solves decoherence,” “extends intrinsic T2,” “accepted by IBM,” “worth $100 billion,” or “proves patent value.” A DO_NOT_CLAIM_PASS result is preserved and used to decide whether observability, timing or physical assumptions failed.

Generalization requires at least three dates and two data qubits, with failed attempts included. A stronger company study should add randomized duplicate circuits, shuffled maps, measurement-error analysis, leakage measurements, T1 controls, neighbor idling, crosstalk monitors and compiler/version locking. Those are not hidden conditions for the first screen; they are the next diligence tier.

19. Security, credentials and reproducibility

The notebook exposes one simple credential line, `key="..."`, because the user requested immediate copy-paste use. It does not call an account-saving API. This minimizes persistence but does not make a published notebook safe: an executed cell can still retain the literal key. The key must be removed, rotated if exposed, and never included in a company package.

Reproducibility depends on more than code. The package stores frozen configuration, environment versions, complete CSVs, compressed audit samples, figures, unit tests and source checksums. The IBM result must additionally store backend name, physical layout, calibrations, transpiled circuits or QPY artifacts where possible, job IDs, UTC times and Qiskit versions.

The simulator and notebook serve different purposes. Local tests verify pure functions and output contracts. Static notebook verification checks syntax and required current-API tokens but cannot substitute for importing Qiskit and transpiling against a live backend. The package labels this limitation instead of presenting an unexecuted notebook as hardware-validated code.

20. Patentability, prior art and ownership

Patent protection cannot be created by strong wording inside a paper. Protection may arise from a filed patent application, enforceable confidentiality, copyright in expression/code, trade-secret practices, or contracts. Each has different requirements. A confidentiality legend is useful notice but may not bind a recipient without agreement. “Patent pending” should not appear until an application has actually been filed.

The field is crowded. Virtual-Z gates, dynamical decoupling, real-time Hamiltonian estimation, predictive compensation, spectator sensing, SMART modulation and closed-loop frequency feedback predate this package [1–8,16–18]. US11695417B1 is particularly relevant to broad closed-loop Ramsey/frequency claims. Counsel should claim no known component in isolation and should prepare element-by-element charts against patents and non-patent literature.

The proposed point of differentiation is the ordered combination: anchor window; two time-ordered multi-axis sensor observations; transition encoding; circular conditional map; support/ambiguity gate; bounded actuation; reserved pilot-validation gate that disables a local map; and matched-latency causal validation. Whether that combination is novel and non-obvious remains unresolved. The invention disclosure and matrix are work inputs, not legal conclusions.

21. Disclosure and filing strategy

CIPO advises inventors to consider filing before disclosure, and grace-period rules differ internationally [19,20]. A Canadian or US grace period is not a safe substitute for a coordinated foreign filing strategy. Before sending non-public implementation details, record the recipient, purpose, exact files, date and NDA status. Public cloud execution may itself create logs; counsel should advise whether that matters for the chosen strategy.

A provisional US application can establish a filing date but must contain adequate technical disclosure and does not mature without further action [21]. It is not a one-page placeholder for an idea. The present package improves enablement by including circuit structure, maps, thresholds, alternatives and failure cases, but counsel must decide jurisdiction, claim scope and inventorship.

Trade-secret protection requires reasonable secrecy measures [22]. If the core map-construction or calibration process is kept secret, distribute only what is necessary under written controls. Conversely, a hardware reproducibility paper may require disclosure. The commercialization decision must therefore choose deliberately between publication, patent filing, confidential evaluation and retained know-how.

22. Commercial evaluation pathway

A technical buyer is unlikely to value an unreplicated simulation at a large figure. A credible path begins with measurable burden and benefit: extra qubits, mid-circuit latency, shots, calibration frequency, compilation complexity and improvement relative to the buyer's best existing suppression. END-QM consumes three sentinels per data qubit in its simplest form, which is expensive for scaling and must be justified by a meaningful effect.

The first commercial milestone is not a sale; it is an independently reproduced evaluation under signed terms. The second is a scoped integration study showing repeatability, resource cost and failure detection. The third is a negotiated option, license, sponsored research agreement or assignment with clear fields of use, milestones, improvements, publication rights, indemnities and background-IP definitions.

No valuation is supplied in this paper. Value depends on effect size, reproducibility, compatibility with error-correction stacks, freedom to operate, alternative solutions and buyer-specific economics. The strongest negotiating asset is a clean, timestamped record of an enabled method and independent data—not an unsupported savings estimate.

23. Engineering roadmap

Phase 1 is the short IBM screen. Phase 2 repeats without changing thresholds on three dates and two data qubits, adds shuffled maps and stores transpiled artifacts. Phase 3 estimates common-mode spatial correlation and tests whether one sentinel bank can serve multiple data qubits. Phase 4 compares END-QM with optimized DD sequences under equal wall-clock time and compiler constraints.

A laboratory branch should test analog END-QM/Dressed only after gate-level observability is established. Its experiments should prospectively sweep modulation rate and amplitude, quantify leakage and heating, and compare equal-power SMART-like controls. A theory branch should derive a noise-transfer function that includes sensor measurement and feedback latency rather than relying only on Monte Carlo forecasts.

A product branch would require online calibration, drift detection, automatic map rollback, audit logs and integration with scheduling. Maps should expire when calibration changes or observability drops. The fallback behavior that appears conservative in simulation becomes a core safety feature in operation.

24. Limitations and failure interpretation

The simulation uses a high common-phase correlation and therefore demonstrates a conditional mechanism rather than a realistic backend forecast. It omits many microscopic effects, including leakage, non-Markovian local noise, readout crosstalk, reset heating, branch-duration variation and correlated gate errors. It treats the selected time grid and T1 envelope simply. Hardware can fail despite correct code.

A positive one-device result can also be misleading. Calibration drift between pilot and confirmation might favor one randomized block; readout asymmetry can bias reconstructed coherence; the virtual-Z convention may interact with analysis phases; and dynamic-circuit scheduling may differ by branch. These risks motivate raw-shot release, randomization, matched controls and replication.

A negative result does not prove that every END concept is false. It can show that this encoder, QPU, timing, resource allocation or common-mode assumption is inadequate. But the preregistered embodiment must still be called failed. Redesign is legitimate only as a new version with a new prospective test.

25. Conclusion

END-QM converts Evans's frame-to-frame radian idea into a bounded and falsifiable quantum-control protocol. Its substantive contribution is not a promise of perfect coherence; it is an experimentally auditable path from two physical sensor frames through a circular transition lexicon to a later data-qubit actuation, with conservative fallback and a matched causal control.

The frozen END-QM-6 simulation passed its model acceptance checks with a 1.222× median AUC ratio and a 1.106× weakest window. The favorable model assumes measurable common-mode drift, so hardware remains the decision point. The included notebook is designed to make that decision quickly while preserving failure evidence.

For company and patent review, the package is intentionally exact about what exists and what does not. It provides enablement, source, tests, outputs, prior-art pressure and conditional result language. It does not manufacture an IBM endorsement, hardware result, valuation or legal monopoly. Those boundaries make a later positive result more credible and a later filing more useful to counsel.

26. Equations and exact algorithm

$$H(t) = -\Delta(t)\sigma_z/2 + \Omega(t)[\cos\varphi_d(t)\sigma_x + \sin\varphi_d(t)\sigma_y]/2 \quad (1)$$

$$\varphi_k = \mathrm{wrap}[\varphi_0 + 2\pi \int_0^{t_k} \delta f(t)\, dt] \quad (2)$$

$$c = s_{k-1} + 8\, s_k, \quad s_k \in \{0,\ldots,7\} \quad (3)$$

$$\mu_c = \mathrm{atan2}(m_{y,c}, m_{x,c}), \quad u_c = \mathrm{clip}(-\mu_c, -\pi/2, +\pi/2) \quad (4)$$

$$C(t) = \sqrt{[\langle X\rangle^2 + \langle Y\rangle^2]}, \quad AUC = \int C(t)/C(0)\, dt \quad (5)$$

$$G_w = AUC_{END,w} / AUC_{matched,w} \quad (6)$$

Algorithm 1 — pilot and confirmation

- Freeze windows, boundary fractions, durations, code-support threshold, correction bound, pilot split and pass rule.
- Collect pilot X/Y shots with two sentinel epochs and no data correction.
- Use pilot-fit shots to estimate conditional circular phase for each window-duration-code cell.
- Set correction to zero for ambiguous/unsupported cells; clip every other angle.
- Evaluate each complete local map on untouched pilot-validation shots; disable it unless gain>1.
- Compile and randomize held-out END, matched-open, wrong-sign, raw-idle and XY4 circuits.
- Calculate normalized equatorial AUC and all-window ratios; issue PASS only under the frozen conjunctive rule.
- Archive raw shots, maps, calibrations, job IDs, versions and failures.

27. Frozen numerical tables

Window	AUC gain	FWER 95% low	FWER 95% high
0	1.3007	1.2949	1.3050
1	1.2878	1.2819	1.2924
2	1.2534	1.2462	1.2594
3	1.2188	1.2111	1.2255
4	1.1579	1.1481	1.1661
5	1.1067	1.0976	1.1151
6	1.1055	1.0968	1.1140
7	1.1086	1.0990	1.1170
8	1.1556	1.1463	1.1640
9	1.2249	1.2171	1.2317
10	1.2613	1.2548	1.2669
11	1.2960	1.2891	1.3014

Table 1. Frozen END-QM-6 held-out synthetic ratios. FWER intervals are simultaneous model-based intervals. They do not quantify backend uncertainty.

Method	Median AUC (µs)
end_qm	96.2255
matched_open	78.7557
oracle_common_phase	157.0142
shuffled_map	75.0863
wrong_sign	75.1235

27.1 Robustness grid

Common scale	Sensor scale	Median correlation	Median gain	Min gain
0.00	0.50	—	1.000	1.000
0.00	1.00	—	1.000	1.000
0.00	2.00	—	1.000	1.000
0.00	4.00	—	1.000	1.000
0.35	0.50	0.992	1.014	1.000
0.35	1.00	0.992	1.011	1.000
0.35	2.00	0.992	1.004	1.000
0.35	4.00	0.992	1.000	1.000
0.60	0.50	0.992	1.073	1.000
0.60	1.00	0.992	1.061	1.000
0.60	2.00	0.992	1.032	1.000
0.60	4.00	0.992	1.000	1.000
1.00	0.50	0.992	1.212	1.111
1.00	1.00	0.992	1.191	1.083
1.00	2.00	0.992	1.141	1.023
1.00	4.00	0.992	1.036	1.000

INTERPRETATION

The favorable frozen setting corresponds to a highly observable common phase. The zero-common rows define a required failure domain. Hardware must measure its own location on this observability continuum.

28. Hardware submission checklist

- [ ] Installation cell completed with Qiskit≈2.5 and qiskit-ibm-runtime≈0.47.
- [ ] API key pasted only in key= and removed before sharing.
- [ ] Backend, four physical qubits, calibration values and durations captured.
- [ ] First transpiled pilot circuit visually inspected.
- [ ] Pilot job ID copied before waiting.
- [ ] Pilot raw CSV and frozen lexicon downloaded.
- [ ] Two confirmation job IDs copied before waiting.
- [ ] Confirmation raw CSV, AUC CSV, decision JSON and plot downloaded.
- [ ] No simulation value inserted into a hardware statement.
- [ ] Failures, missing circuits and backend recalibrations reported.
- [ ] Executed notebook sanitized; key rotated if exposed.
- [ ] Counsel consulted before public disclosure or “patent pending” language.

29. Conditional result language

USE ONLY AFTER JSON STATUS = PASS

On [backend], calibration timestamp [timestamp], the preregistered END-QM-6 screen produced a median normalized equatorial-coherence AUC gain of [value] over the latency-matched dynamic-circuit control, with a paired-window bootstrap 95% interval of [low, high], and outperformed the wrong-sign control. This is a one-device screening result; replication across at least three calibration dates and two data qubits is pending.

USE AFTER ANY NON-PASS

The preregistered END-QM-6 hardware screen did not meet its decision threshold. The result is retained as a falsification or redesign input; no hardware coherence-extension claim is made.

A company cover note should attach raw data and identify the exact notebook commit/checksum. It should not imply that IBM reviewed or accepted the method merely because an IBM QPU executed circuits.

30. References

[1] McKay et al.. Efficient Z gates for quantum computing. Physical Review A 96, 022330 (2017). https://doi.org/10.1103/PhysRevA.96.022330

[2] Cywiński et al.. How to enhance dephasing time in superconducting qubits. Physical Review B 77, 174509 (2008). https://doi.org/10.1103/PhysRevB.77.174509

[3] Hansen et al.. Universal continuous dynamical decoupling via SMART. Physical Review A 104, 062415 (2021). https://doi.org/10.1103/PhysRevA.104.062415

[4] Shulman et al.. Suppressing qubit dephasing using real-time Hamiltonian estimation. Nature Communications 5, 5156 (2014). https://doi.org/10.1038/ncomms6156

[5] Mavadia et al.. Prediction and real-time compensation of qubit decoherence. Nature Communications 8, 14106 (2017). https://doi.org/10.1038/ncomms14106

[6] Singh et al.. Spectator-qubit methods. npj Quantum Information 6, 19 (2020). https://doi.org/10.1038/s41534-020-0251-y

[7] Ezzell et al.. Dynamical decoupling for superconducting qubits: a performance survey. Physical Review Applied 20, 064027 (2023). https://doi.org/10.1103/PhysRevApplied.20.064027

[8] Nature Communications. Real-time two-axis control of a spin qubit. 2024. https://doi.org/10.1038/s41467-024-45857-0

[9] IBM Quantum. Sampler V2 workflow. accessed 2026-08-06. https://quantum.cloud.ibm.com/docs/en/guides/get-started-with-sampler

[10] IBM Quantum. Dynamic circuits. accessed 2026-08-06. https://quantum.cloud.ibm.com/docs/en/guides/execute-dynamic-circuits

[11] IBM Quantum. Plans overview. accessed 2026-08-06. https://quantum.cloud.ibm.com/docs/en/guides/plans-overview

30. References — continued

[12] IBM Quantum. Maximum execution time. accessed 2026-08-06. https://quantum.cloud.ibm.com/docs/en/guides/max-execution-time

[13] IBM Quantum. Estimate job run time. accessed 2026-08-06. https://quantum.cloud.ibm.com/docs/en/guides/estimate-job-run-time

[14] IBM Quantum. Initialize account. accessed 2026-08-06. https://quantum.cloud.ibm.com/docs/en/guides/initialize-account

[15] IBM Quantum. Pulse migration. accessed 2026-08-06. https://quantum.cloud.ibm.com/docs/en/guides/pulse-migration

[16] USPTO. US11695417B1. priority 2022-02-25. https://patents.google.com/patent/US11695417B1/en

[17] USPTO. US20260004174A1. priority 2023-05-23. https://patents.google.com/patent/US20260004174A1/en

[18] WIPO. WO2024259317A1. priority 2023-06-30. https://patents.google.com/patent/WO2024259317A1/en

[19] CIPO. Filing and pre-filing disclosures. accessed 2026-08-06. https://ised-isde.canada.ca/site/canadian-intellectual-property-office/en/corporate-information/canadian-ip-voices-podcast-case-studies-and-blog/filing-patent-application-devil-details

[20] CIPO. Patent grace period consultation. accessed 2026-08-06. https://ised-isde.canada.ca/site/canadian-intellectual-property-office/en/consultation-paper-experiences-patent-grace-period-and-pre-filing-disclosures

[21] USPTO. Provisional application. accessed 2026-08-06. https://www.uspto.gov/patents/basics/apply/provisional-application

[22] WIPO. Trade-secret FAQs. accessed 2026-08-06. https://www.wipo.int/en/web/trade-secrets/tradesecrets_faqs

Appendix A. Simulation core — listing 1 of 9

```
"""Causal END-QM two-frame radian-transition model.

This module models only a correctable phase-noise channel.  It does not model
hardware as a whole and it does not claim to remove T1 relaxation, leakage,
readout error, or unobserved bath entanglement.  Frequencies are in hertz,
times are in seconds, and phases are in radians.
"""

from __future__ import annotations

from dataclasses import asdict, dataclass, replace
from typing import Any

import numpy as np


TAU = 2.0 * np.pi
SENSOR_AXES_RAD = np.array([0.0, TAU / 3.0, 2.0 * TAU / 3.0])


def wrap_angle(value: np.ndarray | float) -> np.ndarray | float:
    """Wrap an angle to [-pi, pi)."""

    return (np.asarray(value) + np.pi) % TAU - np.pi


def raw_transition_code(first_pattern: np.ndarray, second_pattern: np.ndarray) -> np.ndarray:
    """Pack two three-bit sentinel patterns into one six-bit integer."""

    return np.asarray(first_pattern, dtype=np.int16) + 8 * np.asarray(second_pattern,
dtype=np.int16)


def pattern_is_ambiguous(pattern: int) -> bool:
    """All-equal three-sensor outcomes have no preferred hexagonal sector."""

    return int(pattern) in (0, 7)


def transition_is_ambiguous(code: int) -> bool:
    """Return True when either three-bit frame has no sector direction."""

    return pattern_is_ambiguous(code & 7) or pattern_is_ambiguous((code >> 3) & 7)
```

Appendix A. Simulation core — listing 2 of 9

```
0043
0044
0045  def sector_vector(pattern: int) -> complex:
0046      """Return the signed three-axis vector for a sentinel pattern."""
0047
0048      signs = np.array([1.0 if ((pattern >> j) & 1) == 0 else -1.0 for j in range(3)])
0049      return complex(np.sum(signs * np.exp(1j * SENSOR_AXES_RAD)))
0050
0051
0052  def pattern_sector(pattern: int) -> int | None:
0053      """Map a non-ambiguous pattern to one of six radian sectors."""
0054
0055      vector = sector_vector(pattern)
0056      if abs(vector) < 1e-12:
0057          return None
0058      return int(np.round((np.angle(vector) % TAU) / (TAU / 6.0))) % 6
0059
0060
0061  @dataclass(frozen=True)
0062  class ENDQMConfig:
0063      """Frozen synthetic model and confidence-gate parameters."""
0064
0065      seed: int = 20260806
0066      windows: int = 12
0067      times_us: tuple[float, ...] = (0.0, 50.0, 100.0, 150.0, 220.0)
0068      first_boundary_fraction: float = 0.34
0069      second_boundary_fraction: float = 0.70
0070      t1_us: float = 260.0
0071      common_static_sigma_hz: float = 2_080.0
0072      common_slope_sigma_hz_per_s: float = 11_700_000.0
0073      common_sync_sigma_hz: float = 780.0
0074      common_sync_period_us: float = 100.0
0075      data_independent_static_sigma_hz: float = 650.0
0076      data_diffusion_rad_at_220us: float = 0.30
0077      sentinel_independent_sigma_hz: float = 400.0
0078      sentinel_readout_phase_sigma_rad: float = 0.11
0079      concentration_floor: float = 0.10
0080      support_shrinkage: float = 35.0
0081      minimum_cell_support: int = 15
0082      pilot_fit_fraction: float = 0.60
0083      pilot_validation_minimum_gain: float = 1.0
0084      maximum_correction_rad: float = np.pi / 2.0
```

Appendix A. Simulation core — listing 3 of 9

```
    n_train: int = 30_000
    n_test: int = 70_000
    n_bootstrap: int = 6_000

    def to_dict(self) -> dict[str, Any]:
        return asdict(self)


@dataclass(frozen=True)
class LexiconEntry:
    """One raw transition-code entry in the circular residual lexicon."""

    mean_phase_rad: float
    resultant_length: float
    support: int
    confidence: float
    correction_rad: float
    fallback: bool


def _integrated_common_phase(
    time_s: float,
    static_hz: np.ndarray,
    slope_hz_per_s: np.ndarray,
    sync_hz: np.ndarray,
    sync_phase: np.ndarray,
    sync_period_s: float,
) -> np.ndarray:
    """Integrate a static + linear + sinusoidal detuning trajectory."""

    omega = TAU / sync_period_s
    sync_integral = sync_hz / omega * (
        np.cos(sync_phase) - np.cos(omega * time_s + sync_phase)
    )
    return TAU * (
        static_hz * time_s
        + 0.5 * slope_hz_per_s * time_s * time_s
        + sync_integral
    )


def generate_shots(
```

Appendix A. Simulation core — listing 4 of 9

```
    rng: np.random.Generator,
    n_shots: int,
    time_s: float,
    start_phase_rad: float,
    cfg: ENDQMConfig,
    *,
    common_scale: float = 1.0,
    sensor_phase_scale: float = 1.0,
) -> dict[str, np.ndarray]:
    """Generate data phase and two measured three-sentinel frames.

    The same latent detuning drives the data qubit and both sentinel frames.
    Independent data and sentinel terms make transfer imperfect.  The model is
    intentionally favorable to a common-mode tracker; robustness sweeps expose
    where that assumption fails.
    """

    static = rng.normal(0.0, cfg.common_static_sigma_hz * common_scale, n_shots)
    slope = rng.normal(0.0, cfg.common_slope_sigma_hz_per_s * common_scale, n_shots)
    sync = rng.normal(0.0, cfg.common_sync_sigma_hz * common_scale, n_shots)
    sync_phase = rng.uniform(0.0, TAU, n_shots)
    period_s = cfg.common_sync_period_us * 1e-6

    def common_at(fraction: float) -> np.ndarray:
        return _integrated_common_phase(
            time_s * fraction, static, slope, sync, sync_phase, period_s
        )

    common_total = common_at(1.0)
    diffusion_sigma = cfg.data_diffusion_rad_at_220us * np.sqrt(
        max(time_s, 0.0) / 220e-6
    )
    data_phase = (
        common_total
        + TAU
        * rng.normal(0.0, cfg.data_independent_static_sigma_hz, n_shots)
        * time_s
        + rng.normal(0.0, diffusion_sigma, n_shots)
        + start_phase_rad
    )

    patterns: list[np.ndarray] = []
```

Appendix A. Simulation core — listing 5 of 9

```
    sentinel_latent: list[np.ndarray] = []
    for fraction in (cfg.first_boundary_fraction, cfg.second_boundary_fraction):
        boundary_s = time_s * fraction
        common_phase = common_at(fraction)
        phase = (
            common_phase[:, None]
            + start_phase_rad
            + TAU
            * rng.normal(
                0.0,
                cfg.sentinel_independent_sigma_hz * sensor_phase_scale,
                size=(n_shots, 3),
            )
            * boundary_s
            + rng.normal(
                0.0,
                cfg.sentinel_readout_phase_sigma_rad * sensor_phase_scale,
                size=(n_shots, 3),
            )
        )
        probability_zero = (1.0 + np.cos(phase - SENSOR_AXES_RAD[None, :])) / 2.0
        bits = (rng.random((n_shots, 3)) > probability_zero).astype(np.int8)
        patterns.append(bits[:, 0] + 2 * bits[:, 1] + 4 * bits[:, 2])
        sentinel_latent.append(common_phase)

    code = raw_transition_code(patterns[0], patterns[1])
    return {
        "data_phase_rad": data_phase,
        "transition_code": code,
        "first_pattern": patterns[0],
        "second_pattern": patterns[1],
        "common_total_phase_rad": common_total,
        "second_sentinel_latent_phase_rad": sentinel_latent[1],
    }


def fit_transition_lexicon(
    data_phase_rad: np.ndarray,
    transition_code: np.ndarray,
    cfg: ENDQMConfig,
    *,
    zero_correction: bool = False,
```

Appendix A. Simulation core — listing 6 of 9

```
) -> tuple[np.ndarray, list[LexiconEntry]]:
    """Fit a training-only 64-code circular lexicon with automatic fallback."""

    corrections = np.zeros(64, dtype=float)
    entries: list[LexiconEntry] = []
    for code in range(64):
        values = data_phase_rad[transition_code == code]
        support = int(values.size)
        if support:
            circular_mean = np.mean(np.exp(1j * values))
            mu = float(np.angle(circular_mean))
            resultant = float(abs(circular_mean))
        else:
            mu = 0.0
            resultant = 0.0
        concentration = np.clip(
            (resultant - cfg.concentration_floor)
            / max(1.0 - cfg.concentration_floor, 1e-12),
            0.0,
            1.0,
        )
        support_factor = np.sqrt(support / (support + cfg.support_shrinkage)) if support else 0.0
        confidence = float(concentration * support_factor)
        fallback = (
            zero_correction
            or support < cfg.minimum_cell_support
            or transition_is_ambiguous(code)
        )
        correction = 0.0 if fallback else float(
            np.clip(-confidence * mu, -cfg.maximum_correction_rad, cfg.maximum_correction_rad)
        )
        corrections[code] = correction
        entries.append(
            LexiconEntry(
                mean_phase_rad=mu,
                resultant_length=resultant,
                support=support,
                confidence=confidence,
                correction_rad=correction,
                fallback=fallback,
            )
        )
```

Appendix A. Simulation core — listing 7 of 9

```
    return corrections, entries


def _coherence_stats(phase_rad: np.ndarray, t1_envelope: float) -> dict[str, float]:
    values = np.column_stack((np.cos(phase_rad), np.sin(phase_rad)))
    mean = values.mean(axis=0)
    cov = np.cov(values, rowvar=False, ddof=1) / max(len(values), 1)
    return {
        "x": float(mean[0] * t1_envelope),
        "y": float(mean[1] * t1_envelope),
        "coherence": float(np.hypot(*mean) * t1_envelope),
        "cov_xx": float(cov[0, 0] * t1_envelope**2),
        "cov_xy": float(cov[0, 1] * t1_envelope**2),
        "cov_yy": float(cov[1, 1] * t1_envelope**2),
        "shots": int(len(values)),
    }


def simulate_window(
    rng: np.random.Generator,
    time_us: float,
    window_index: int,
    cfg: ENDQMConfig,
    *,
    common_scale: float = 1.0,
    sensor_phase_scale: float = 1.0,
    n_train: int | None = None,
    n_test: int | None = None,
) -> dict[str, Any]:
    """Fit on pilot shots and evaluate independent held-out shots."""

    time_s = time_us * 1e-6
    start_phase = TAU * window_index / cfg.windows
    train = generate_shots(
        rng,
        n_train or cfg.n_train,
        time_s,
        start_phase,
        cfg,
        common_scale=common_scale,
        sensor_phase_scale=sensor_phase_scale,
    )
```

Appendix A. Simulation core — listing 8 of 9

```
    split = int(len(train["data_phase_rad"]) * cfg.pilot_fit_fraction)
    corrections, entries = fit_transition_lexicon(
        train["data_phase_rad"][:split],
        train["transition_code"][:split],
        cfg,
        zero_correction=(time_us == 0.0),
    )
    validation_phase = train["data_phase_rad"][split:]
    validation_code = train["transition_code"][split:]
    validation_open = abs(np.mean(np.exp(1j * validation_phase)))
    validation_end = abs(
        np.mean(np.exp(1j * (validation_phase + corrections[validation_code])))
    )
    validation_gain = float(validation_end / max(validation_open, 1e-12))
    pilot_gate_pass = bool(
        time_us > 0.0 and validation_gain > cfg.pilot_validation_minimum_gain
    )
    if not pilot_gate_pass:
        corrections = np.zeros_like(corrections)
        entries = [
            replace(entry, correction_rad=0.0, fallback=True) for entry in entries
        ]
    test = generate_shots(
        rng,
        n_test or cfg.n_test,
        time_s,
        start_phase,
        cfg,
        common_scale=common_scale,
        sensor_phase_scale=sensor_phase_scale,
    )
    code = test["transition_code"]
    data = test["data_phase_rad"]
    shuffled = np.roll(corrections, 8)
    t1_envelope = float(np.exp(-time_us / (2.0 * cfg.t1_us)))
    methods = {
        "matched_open": data,
        "end_qm": data + corrections[code],
        "wrong_sign": data - corrections[code],
        "shuffled_map": data + shuffled[code],
        "oracle_common_phase": data - test["common_total_phase_rad"],
    }
```

Appendix A. Simulation core — listing 9 of 9

```
    stats = {name: _coherence_stats(phase, t1_envelope) for name, phase in methods.items()}
    base_phase = methods["matched_open"]
    base_vectors = np.column_stack((np.cos(base_phase), np.sin(base_phase))) * t1_envelope
    for name, phase in methods.items():
        method_vectors = np.column_stack((np.cos(phase), np.sin(phase))) * t1_envelope
        joint = np.column_stack((method_vectors, base_vectors))
        joint_cov = np.cov(joint, rowvar=False, ddof=1) / max(len(joint), 1)
        for row in range(4):
            for col in range(4):
                stats[name][f"paired_cov_{row}{col}"] = float(joint_cov[row, col])
        stats[name]["identical_to_matched"] = bool(
            np.array_equal(phase, base_phase)
        )
    true_phase = test["common_total_phase_rad"]
    sentinel_phase = test["second_sentinel_latent_phase_rad"]
    if np.std(true_phase) < 1e-15 or np.std(sentinel_phase) < 1e-15:
        corr = float("nan")
    else:
        corr = float(np.corrcoef(true_phase, sentinel_phase)[0, 1])
    fallback_fraction = float(np.mean([entry.fallback for entry in entries]))
    return {
        "time_us": float(time_us),
        "window": int(window_index),
        "start_phase_rad": float(start_phase),
        "methods": stats,
        "corrections_rad": corrections,
        "lexicon_entries": entries,
        "latent_phase_correlation": corr,
        "fallback_code_fraction": fallback_fraction,
        "pilot_validation_gain": validation_gain,
        "pilot_gate_pass": pilot_gate_pass,
        "audit_sample": {
            "data_phase_rad": data[:512],
            "transition_code": code[:512],
        },
    }
```

Appendix B. Frozen simulation runner — listing 1 of 10

```
#!/usr/bin/env python3
"""Run the frozen END-QM synthetic validation and robustness suite."""

from __future__ import annotations

import argparse
import csv
import gzip
import json
import platform
import sys
from dataclasses import asdict
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.stats import norm

from endqm.core import ENDQMConfig, simulate_window


METHOD_LABELS = {
    "matched_open": "Matched open-loop",
    "end_qm": "END-QM",
    "wrong_sign": "Wrong sign",
    "shuffled_map": "Shuffled map",
    "oracle_common_phase": "Common-phase oracle",
}


def write_json(path: Path, value: object) -> None:
    path.parent.mkdir(parents=True, exist_ok=True)
    path.write_text(json.dumps(value, indent=2), encoding="utf-8")


def run_full(cfg: ENDQMConfig, out_dir: Path) -> tuple[pd.DataFrame, pd.DataFrame, dict]:
    rng = np.random.default_rng(cfg.seed)
    rows: list[dict] = []
    lexicons: list[dict] = []
    audit_rows: list[dict] = []
    for window in range(cfg.windows):
```

Appendix B. Frozen simulation runner — listing 2 of 10

```
0043          for time_us in cfg.times_us:
0044              result = simulate_window(rng, time_us, window, cfg)
0045              for method, stats in result["methods"].items():
0046                  rows.append(
0047                      {
0048                          "window": window,
0049                          "start_phase_rad": result["start_phase_rad"],
0050                          "time_us": time_us,
0051                          "method": method,
0052                          **stats,
0053                          "latent_phase_correlation": result["latent_phase_correlation"],
0054                          "fallback_code_fraction": result["fallback_code_fraction"],
0055                          "pilot_validation_gain": result["pilot_validation_gain"],
0056                          "pilot_gate_pass": result["pilot_gate_pass"],
0057                      }
0058                  )
0059              lexicons.append(
0060                  {
0061                      "window": window,
0062                      "time_us": time_us,
0063                      "entries": [asdict(entry) for entry in result["lexicon_entries"]],
0064                  }
0065              )
0066              audit = result["audit_sample"]
0067              for phase, code in zip(audit["data_phase_rad"], audit["transition_code"]):
0068                  audit_rows.append(
0069                      {
0070                          "window": window,
0071                          "time_us": time_us,
0072                          "data_phase_rad": float(phase),
0073                          "transition_code": int(code),
0074                      }
0075                  )
0076
0077      curves = pd.DataFrame(rows)
0078      curves.to_csv(out_dir / "dynamic_coherence_stats.csv", index=False)
0079      pd.DataFrame(audit_rows).to_csv(
0080          out_dir / "dynamic_audit_phase_samples.csv.gz", index=False, compression="gzip"
0081      )
0082      with gzip.open(out_dir / "dynamic_lexicon_entries.json.gz", "wt", encoding="utf-8") as
handle:
0083          json.dump(lexicons, handle)
0084
```

Appendix B. Frozen simulation runner — listing 3 of 10

```
    auc_rows: list[dict] = []
    for (window, method), group in curves.groupby(["window", "method"]):
        group = group.sort_values("time_us")
        auc_rows.append(
            {
                "window": int(window),
                "method": str(method),
                "auc_us": float(np.trapezoid(group.coherence, group.time_us)),
            }
        )
    auc = pd.DataFrame(auc_rows)
    auc.to_csv(out_dir / "dynamic_auc_by_window.csv", index=False)

    # Conditional parametric bootstrap of held-out shot means. The pilot map is
    # held frozen, matching the intended pilot/confirmation hardware split.
    bootstrap_rng = np.random.default_rng(cfg.seed + 11)
    boot_auc: dict[tuple[int, str], np.ndarray] = {}
    boot_gain: dict[tuple[int, str], np.ndarray] = {}
    for (window, method), group in curves.groupby(["window", "method"]):
        group = group.sort_values("time_us")
        coherence_draws = []
        matched_draws = []
        for row in group.itertuples(index=False):
            if bool(row.identical_to_matched):
                cov = np.array([[row.cov_xx, row.cov_xy], [row.cov_xy, row.cov_yy]])
                cov += np.eye(2) * 1e-15
                draws = bootstrap_rng.multivariate_normal(
                    np.array([row.x, row.y]), cov, size=cfg.n_bootstrap
                )
                coherence = np.clip(np.hypot(draws[:, 0], draws[:, 1]), 0.0, 1.0)
                coherence_draws.append(coherence)
                matched_draws.append(coherence)
            else:
                covariance = np.array(
                    [
                        [getattr(row, f"paired_cov_{r}{c}") for c in range(4)]
                        for r in range(4)
                    ]
                )
                covariance += np.eye(4) * 1e-15
                base_row = curves[
                    (curves.window == row.window)
```

Appendix B. Frozen simulation runner — listing 4 of 10

```
                    & (curves.time_us == row.time_us)
                    & (curves.method == "matched_open")
                ].iloc[0]
                mean = np.array([row.x, row.y, base_row.x, base_row.y])
                draws = bootstrap_rng.multivariate_normal(
                    mean, covariance, size=cfg.n_bootstrap
                )
                coherence_draws.append(
                    np.clip(np.hypot(draws[:, 0], draws[:, 1]), 0.0, 1.0)
                )
                matched_draws.append(
                    np.clip(np.hypot(draws[:, 2], draws[:, 3]), 0.0, 1.0)
                )
        boot_auc[(int(window), str(method))] = np.trapezoid(
            np.vstack(coherence_draws), group.time_us.to_numpy(), axis=0
        )
        paired_base_auc = np.trapezoid(
            np.vstack(matched_draws), group.time_us.to_numpy(), axis=0
        )
        boot_gain[(int(window), str(method))] = boot_auc[(int(window), str(method))] / np.clip(
            paired_base_auc, 1e-12, None
        )

    point = auc.pivot(index="window", columns="method", values="auc_us")
    gain_rows: list[dict] = []
    family_tail = 0.05 / (2.0 * cfg.windows)
    for window in range(cfg.windows):
        for method in ["end_qm", "wrong_sign", "shuffled_map", "oracle_common_phase"]:
            samples = boot_gain[(window, method)]
            gain_rows.append(
                {
                    "window": window,
                    "method": method,
                    "auc_gain_vs_matched": float(
                        point.loc[window, method] / point.loc[window, "matched_open"]
                    ),
                    "fwer95_low": float(np.quantile(samples, family_tail)),
                    "fwer95_high": float(np.quantile(samples, 1.0 - family_tail)),
                }
            )
    gains = pd.DataFrame(gain_rows)
    gains.to_csv(out_dir / "dynamic_gain_by_window.csv", index=False)
```

Appendix B. Frozen simulation runner — listing 5 of 10

```
0169
0170      end_gain = gains[gains.method == "end_qm"]
0171      med_boot = np.median(
0172          np.vstack(
0173              [
0174                  boot_gain[(window, "end_qm")]
0175                  for window in range(cfg.windows)
0176              ]
0177          ),
0178          axis=0,
0179      )
0180      summary = {
0181          "status": "synthetic_held_out_forecast_only",
0182          "protocol": "END-QM two-frame six-sector spectator feedback",
0183          "seed": cfg.seed,
0184          "windows": cfg.windows,
0185          "times_us": list(cfg.times_us),
0186          "n_train_per_window_time": cfg.n_train,
0187          "n_test_per_window_time": cfg.n_test,
0188          "median_auc_gain_vs_matched": float(end_gain.auc_gain_vs_matched.median()),
0189          "median_auc_gain_ci95": [
0190              float(np.quantile(med_boot, 0.025)),
0191              float(np.quantile(med_boot, 0.975)),
0192          ],
0193          "minimum_window_auc_gain": float(end_gain.auc_gain_vs_matched.min()),
0194          "minimum_window_fwer95_low": float(end_gain.fwer95_low.min()),
0195          "all_windows_point_gain_gt_1": bool((end_gain.auc_gain_vs_matched > 1.0).all()),
0196          "all_windows_fwer95_low_gt_1": bool((end_gain.fwer95_low > 1.0).all()),
0197          "all_windows_point_gain_gte_1": bool((end_gain.auc_gain_vs_matched >= 1.0).all()),
0198          "all_windows_fwer95_low_gte_1": bool((end_gain.fwer95_low >= 1.0).all()),
0199          "median_latent_phase_correlation": float(curves.latent_phase_correlation.median()),
0200          "median_fallback_code_fraction": float(curves.fallback_code_fraction.median()),
0201          "method_median_auc_us": {
0202              method: float(values.median())
0203              for method, values in auc.groupby("method").auc_us
0204          },
0205          "claim_boundary": (
0206              "This is a synthetic forecast conditioned on common-mode measurable phase drift. "
0207              "It is not an IBM hardware result and does not establish intrinsic T2 extension."
0208          ),
0209      }
0210      write_json(out_dir / "dynamic_summary.json", summary)
```

Appendix B. Frozen simulation runner — listing 6 of 10

```
    return curves, gains, summary


def run_robustness(cfg: ENDQMConfig, out_dir: Path) -> pd.DataFrame:
    rows: list[dict] = []
    common_scales = [0.0, 0.35, 0.60, 1.0, 1.30, 1.60]
    sensor_scales = [0.5, 1.0, 2.0, 4.0]
    for i, common_scale in enumerate(common_scales):
        for j, sensor_scale in enumerate(sensor_scales):
            rng = np.random.default_rng(cfg.seed + 1000 + i * 100 + j)
            auc_by_method: dict[str, list[float]] = {"matched_open": [], "end_qm": []}
            correlations: list[float] = []
            for window in range(cfg.windows):
                curves = {"matched_open": [], "end_qm": []}
                for time_us in cfg.times_us:
                    result = simulate_window(
                        rng,
                        time_us,
                        window,
                        cfg,
                        common_scale=common_scale,
                        sensor_phase_scale=sensor_scale,
                        n_train=5_000,
                        n_test=12_000,
                    )
                    correlations.append(result["latent_phase_correlation"])
                    for method in curves:
                        curves[method].append(result["methods"][method]["coherence"])
                for method in curves:
                    auc_by_method[method].append(
                        float(np.trapezoid(curves[method], cfg.times_us))
                    )
            gains = np.asarray(auc_by_method["end_qm"]) / np.clip(
                np.asarray(auc_by_method["matched_open"]), 1e-12, None
            )
            finite_correlations = [value for value in correlations if np.isfinite(value)]
            rows.append(
                {
                    "common_scale": common_scale,
                    "sensor_noise_scale": sensor_scale,
                    "median_latent_phase_correlation": (
                        float(np.median(finite_correlations))
```

Appendix B. Frozen simulation runner — listing 7 of 10

```
                        if finite_correlations
                        else None
                    ),
                    "median_auc_gain": float(np.median(gains)),
                    "minimum_window_auc_gain": float(np.min(gains)),
                    "all_windows_gain_gt_1": bool(np.all(gains > 1.0)),
                }
            )
    result = pd.DataFrame(rows)
    result.to_csv(out_dir / "dynamic_robustness_sweep.csv", index=False)
    return result


def make_figures(
    curves: pd.DataFrame,
    gains: pd.DataFrame,
    robustness: pd.DataFrame,
    out_dir: Path,
) -> None:
    fig_dir = out_dir / "figures"
    fig_dir.mkdir(parents=True, exist_ok=True)
    plt.rcParams.update({"font.size": 10, "axes.titlesize": 12, "axes.labelsize": 10})

    fig, ax = plt.subplots(figsize=(7.3, 4.5))
    for method in ["matched_open", "end_qm", "wrong_sign", "oracle_common_phase"]:
        group = curves[curves.method == method].groupby("time_us").coherence
        mean = group.median()
        low = group.quantile(0.10)
        high = group.quantile(0.90)
        ax.plot(mean.index, mean.values, marker="o", label=METHOD_LABELS[method])
        ax.fill_between(mean.index, low.values, high.values, alpha=0.10)
    ax.set(xlabel="Nominal storage duration (us)", ylabel="Equatorial coherence", ylim=(0, 1.03))
    ax.set_title("Held-out END-QM dynamic-sentinel forecast")
    ax.grid(alpha=0.25)
    ax.legend(frameon=False, fontsize=8)
    fig.tight_layout()
    fig.savefig(fig_dir / "fig_dynamic_coherence.png", dpi=240)
    plt.close(fig)

    end = gains[gains.method == "end_qm"].sort_values("window")
    fig, ax = plt.subplots(figsize=(7.3, 4.3))
    yerr = np.vstack(
```

Appendix B. Frozen simulation runner — listing 8 of 10

```
            [
                end.auc_gain_vs_matched - end.fwer95_low,
                end.fwer95_high - end.auc_gain_vs_matched,
            ]
        )
        ax.errorbar(end.window, end.auc_gain_vs_matched, yerr=yerr, fmt="o", capsize=3)
        ax.axhline(1.0, color="black", linewidth=1, linestyle="--")
        ax.set(
            xlabel="Pre-registered radian window",
            ylabel="AUC gain vs matched open-loop",
            title="All-window held-out END-QM gain",
            xticks=range(12),
        )
        ax.grid(alpha=0.2)
        fig.tight_layout()
        fig.savefig(fig_dir / "fig_dynamic_window_gain.png", dpi=240)
        plt.close(fig)

        auc_median = (
            gains.groupby("method").auc_gain_vs_matched.median().reindex(
                ["end_qm", "wrong_sign", "shuffled_map", "oracle_common_phase"]
            )
        )
        fig, ax = plt.subplots(figsize=(7.3, 4.2))
        bars = ax.bar(
            [METHOD_LABELS[index] for index in auc_median.index], auc_median.values
        )
        ax.axhline(1.0, color="black", linewidth=1, linestyle="--")
        ax.set(ylabel="Median AUC gain vs matched open-loop", title="Specificity and upper-bound
controls")
        ax.tick_params(axis="x", rotation=18)
        for bar, value in zip(bars, auc_median.values):
            ax.text(bar.get_x() + bar.get_width() / 2, value + 0.02, f"{value:.2f}x", ha="center",
fontsize=9)
        fig.tight_layout()
        fig.savefig(fig_dir / "fig_dynamic_controls.png", dpi=240)
        plt.close(fig)

        pivot = robustness.pivot(
            index="sensor_noise_scale", columns="common_scale", values="median_auc_gain"
        )
        fig, ax = plt.subplots(figsize=(7.3, 4.4))
        image = ax.imshow(pivot.values, aspect="auto", origin="lower", cmap="viridis")
        ax.set_xticks(range(len(pivot.columns)), [f"{v:g}" for v in pivot.columns])
```

Appendix B. Frozen simulation runner — listing 9 of 10

```
0337      ax.set_yticks(range(len(pivot.index)), [f"{v:g}" for v in pivot.index])
0338      ax.set(xlabel="Common-mode phase scale", ylabel="Sentinel noise scale", title="END-QM
robustness operating map")
0339      for row in range(pivot.shape[0]):
0340          for col in range(pivot.shape[1]):
0341              value = pivot.iloc[row, col]
0342              ax.text(col, row, f"{value:.2f}", ha="center", va="center", color="white" if value <
1.2 else "black", fontsize=8)
0343      fig.colorbar(image, ax=ax, label="Median AUC gain")
0344      fig.tight_layout()
0345      fig.savefig(fig_dir / "fig_dynamic_robustness.png", dpi=240)
0346      plt.close(fig)
0347
0348      # Exact six-sector geometry used by the three-sentinel code.
0349      angles = np.arange(6) * 2 * np.pi / 6
0350      fig, ax = plt.subplots(figsize=(5.6, 5.3), subplot_kw={"projection": "polar"})
0351      ax.scatter(angles, np.ones(6), s=80)
0352      for sector, angle in enumerate(angles):
0353          ax.text(angle, 1.12, f"S{sector}", ha="center", va="center")
0354      ax.set_rticks([])
0355      ax.set_ylim(0, 1.25)
0356      ax.set_title("Six resolvable radian sectors from three 120-degree sentinels", pad=20)
0357      fig.tight_layout()
0358      fig.savefig(fig_dir / "fig_six_sector_code.png", dpi=240)
0359      plt.close(fig)
0360
0361
0362  def main() -> None:
0363      parser = argparse.ArgumentParser()
0364      parser.add_argument("--output-dir", type=Path, required=True)
0365      args = parser.parse_args()
0366      args.output_dir.mkdir(parents=True, exist_ok=True)
0367      cfg = ENDQMConfig()
0368      write_json(args.output_dir / "frozen_dynamic_config.json", cfg.to_dict())
0369      curves, gains, summary = run_full(cfg, args.output_dir)
0370      robustness = run_robustness(cfg, args.output_dir)
0371      make_figures(curves, gains, robustness, args.output_dir)
0372      write_json(
0373          args.output_dir / "environment.json",
0374          {
0375              "python": sys.version,
0376              "platform": platform.platform(),
0377              "numpy": np.__version__,
0378              "pandas": pd.__version__,
```

Appendix B. Frozen simulation runner — listing 10 of 10

```
            "scipy": sys.modules["scipy"].__version__,
            "matplotlib": sys.modules["matplotlib"].__version__,
        },
    )
    print(json.dumps(summary, indent=2))


if __name__ == "__main__":
    main()
```

Appendix C. Unit tests — listing 1 of 2

```
import unittest

import numpy as np

from endqm.core import (
    ENDQMConfig,
    fit_transition_lexicon,
    pattern_sector,
    raw_transition_code,
    simulate_window,
    transition_is_ambiguous,
    wrap_angle,
)


class CoreTests(unittest.TestCase):
    def test_wrap_range(self):
        values = wrap_angle(np.linspace(-20.0, 20.0, 1001))
        self.assertTrue(np.all(values >= -np.pi))
        self.assertTrue(np.all(values < np.pi))

    def test_six_non_ambiguous_patterns_map_to_six_sectors(self):
        sectors = [pattern_sector(pattern) for pattern in range(8)]
        self.assertIsNone(sectors[0])
        self.assertIsNone(sectors[7])
        self.assertEqual(set(s for s in sectors if s is not None), set(range(6)))

    def test_raw_code_packing(self):
        first = np.array([0, 1, 7])
        second = np.array([7, 2, 0])
        np.testing.assert_array_equal(raw_transition_code(first, second), [56, 17, 7])

    def test_ambiguous_frames_force_fallback(self):
        for code in range(64):
            expected = ((code & 7) in (0, 7)) or (((code >> 3) & 7) in (0, 7))
            self.assertEqual(transition_is_ambiguous(code), expected)

    def test_lexicon_is_bounded_and_support_gated(self):
        cfg = ENDQMConfig(minimum_cell_support=15)
        phase = np.full(20, 1.2)
        code = np.full(20, 9)
        corrections, entries = fit_transition_lexicon(phase, code, cfg)
```

Appendix C. Unit tests — listing 2 of 2

```
0043          self.assertLessEqual(abs(corrections[9]), np.pi)
0044          self.assertFalse(entries[9].fallback)
0045          self.assertTrue(entries[10].fallback)
0046
0047      def test_frozen_specificity_direction(self):
0048          cfg = ENDQMConfig(n_train=5000, n_test=15000)
0049          result = simulate_window(np.random.default_rng(1234), 100.0, 4, cfg)
0050          self.assertGreater(
0051              result["methods"]["end_qm"]["coherence"],
0052              result["methods"]["matched_open"]["coherence"],
0053          )
0054          self.assertGreater(
0055              result["methods"]["end_qm"]["coherence"],
0056              result["methods"]["wrong_sign"]["coherence"],
0057          )
0058
0059
0060  if __name__ == "__main__":
0061      unittest.main()
0062
```

Appendix D. IBM 10-minute Colab code — listing 1 of 10

```
# --- Notebook code cell 2 ---
%pip -q install "qiskit~=2.5.0" "qiskit-ibm-runtime~=0.47.0" pandas matplotlib scipy

# --- Notebook code cell 4 ---
key="PASTE_YOUR_IBM_CLOUD_API_KEY_HERE"

if key == "PASTE_YOUR_IBM_CLOUD_API_KEY_HERE" or len(key) < 20:
    raise ValueError("Paste your IBM Cloud API key into key= before continuing.")

# --- Notebook code cell 5 ---
from __future__ import annotations

import json, math, random, sys, time
from dataclasses import dataclass
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.stats import bootstrap

from qiskit import ClassicalRegister, QuantumCircuit, QuantumRegister, transpile
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2 as Sampler

SEED = 20260806
random.seed(SEED)
np.random.seed(SEED)

service = QiskitRuntimeService(token=key, instance="auto")
print("Authenticated. The API key is held only in notebook memory.")

# --- Notebook code cell 7 ---
BACKEND_NAME = None  # Optional: set to a specific QPU name, otherwise least-busy is used.

if BACKEND_NAME:
    backend = service.backend(BACKEND_NAME)
else:
    backend = service.least_busy(operational=True, simulator=False, min_num_qubits=4)

props = backend.properties()
rows = []
```

Appendix D. IBM 10-minute Colab code — listing 2 of 10

```
for q in range(backend.num_qubits):
    try:
        t1 = float(props.t1(q))
        t2 = float(props.t2(q))
        readout_error = float(props.readout_error(q))
        score = min(t2, 2.0 * t1) / max(0.01, readout_error)
        rows.append({"qubit": q, "t1_s": t1, "t2_s": t2,
                     "readout_error": readout_error, "score": score})
    except Exception:
        pass

calibration = pd.DataFrame(rows).sort_values("score", ascending=False)
if len(calibration) < 4:
    raise RuntimeError("Could not find four qubits with complete calibration data.")

physical_qubits = calibration.head(4).qubit.astype(int).tolist()
data_physical, *sentinel_physical = physical_qubits
data_t2_us = float(calibration.iloc[0].t2_s * 1e6)
durations_us = sorted(set([0.0] + [
    round(float(np.clip(frac * data_t2_us, 16.0, 220.0)), 1)
    for frac in (0.25, 0.50, 0.85)
]))
while len(durations_us) < 4:
    durations_us = sorted(set(durations_us + [float(16 * len(durations_us))]))

print("Backend:", backend.name)
print("Physical layout [data, sentinel 0, 1, 2]:", physical_qubits)
print("Data T2 from current calibration (us):", round(data_t2_us, 2))
print("Test durations (us):", durations_us)
display(calibration.head(10))

# --- Notebook code cell 9 ---
WINDOWS = np.linspace(0.0, 2*np.pi, 12, endpoint=False)
SENTINEL_AXES = (0.0, 2*np.pi/3, 4*np.pi/3)
BOUNDARIES = (0.34, 0.70)
PILOT_SHOTS = 2048
CONFIRM_SHOTS = 512
MAX_CORRECTION_RAD = np.pi/2
MIN_CODE_SUPPORT = 32
METHODS = ("end_qm", "matched_open", "wrong_sign", "raw_idle", "xy4")
AMBIGUOUS_HALF_CODES = {0b000, 0b111}
PILOT_JOB_CAP_S = 160
```

Appendix D. IBM 10-minute Colab code — listing 3 of 10

```
CONFIRM_JOB_CAP_S = 190  # two confirmation jobs: 160 + 190 + 190 = 540 seconds

def wrap(x):
    return (np.asarray(x) + np.pi) % (2*np.pi) - np.pi

def code_is_ambiguous(code_value):
    return ((code_value & 0b111) in AMBIGUOUS_HALF_CODES or
            (((code_value >> 3) & 0b111) in AMBIGUOUS_HALF_CODES))

def add_data_analysis(qc, q, axis):
    if axis == "x":
        qc.h(q)
    elif axis == "y":
        qc.sdg(q); qc.h(q)
    else:
        raise ValueError(axis)

def add_sentinel_analysis(qc, q, axis):
    qc.rz(-axis, q); qc.h(q)

def base_registers():
    q = QuantumRegister(4, "q")
    sense = ClassicalRegister(6, "sense")
    readout = ClassicalRegister(1, "readout")
    return q, sense, readout, QuantumCircuit(q, sense, readout)

def build_dynamic_circuit(window, duration_us, axis, correction_map=None, sign=1.0):
    q, sense, readout, qc = base_registers()
    data = q[0]; sentinels = [q[1], q[2], q[3]]
    qc.h(data); qc.rz(float(window), data)
    for s in sentinels:
        qc.h(s); qc.rz(float(window), s)

    cuts = [BOUNDARIES[0]*duration_us,
            (BOUNDARIES[1]-BOUNDARIES[0])*duration_us,
            (1-BOUNDARIES[1])*duration_us]
    for qb in q:
        qc.delay(max(cuts[0], 0.0), qb, unit="us")
    qc.barrier()
    for i, (s, a) in enumerate(zip(sentinels, SENTINEL_AXES)):
        add_sentinel_analysis(qc, s, a)
        qc.measure(s, sense[i])
```

Appendix D. IBM 10-minute Colab code — listing 4 of 10

```
    qc.barrier()
    for s in sentinels:
        qc.reset(s); qc.h(s); qc.rz(float(window), s)

    for qb in q:
        qc.delay(max(cuts[1], 0.0), qb, unit="us")
    qc.barrier()
    for i, (s, a) in enumerate(zip(sentinels, SENTINEL_AXES)):
        add_sentinel_analysis(qc, s, a)
        qc.measure(s, sense[i+3])
    qc.barrier()

    if correction_map:
        for raw_code in range(64):
            angle = float(sign * correction_map.get(raw_code, 0.0))
            if abs(angle) > 1e-12:
                with qc.if_test((sense, raw_code)):
                    qc.rz(angle, data)

    for qb in q:
        qc.delay(max(cuts[2], 0.0), qb, unit="us")
    qc.rz(float(-window), data)
    add_data_analysis(qc, data, axis)
    qc.measure(data, readout[0])
    return qc

def build_reference_circuit(window, duration_us, axis, method):
    q, sense, readout, qc = base_registers()
    data = q[0]
    qc.h(data); qc.rz(float(window), data)
    if method == "xy4" and duration_us > 0:
        for pulse in ("x", "y", "x", "y"):
            qc.delay(duration_us/5, data, unit="us")
            getattr(qc, pulse)(data)
        qc.delay(duration_us/5, data, unit="us")
    else:
        qc.delay(duration_us, data, unit="us")
    qc.rz(float(-window), data)
    add_data_analysis(qc, data, axis)
    qc.measure(data, readout[0])
    return qc

```

Appendix D. IBM 10-minute Colab code — listing 5 of 10

```
def manifest_key(window_index, duration_index, axis, method):
    return f"w{window_index:02d}_t{duration_index:02d}_{axis}_{method}"

# --- Notebook code cell 11 ---
pilot_manifest = []
pilot_circuits = []
for wi, window in enumerate(WINDOWS):
    for ti, duration in enumerate(durations_us):
        for axis in ("x", "y"):
            pilot_manifest.append({"key": manifest_key(wi, ti, axis, "pilot"),
                                   "window_index": wi, "duration_index": ti,
                                   "window_rad": float(window), "duration_us": float(duration),
                                   "axis": axis, "method": "pilot"})
            pilot_circuits.append(build_dynamic_circuit(window, duration, axis))

assert len(pilot_circuits) == 96
pass_manager = generate_preset_pass_manager(
    backend=backend, optimization_level=1, initial_layout=physical_qubits,
    seed_transpiler=SEED,
)
pilot_isa = pass_manager.run(pilot_circuits)
print("Pilot circuits:", len(pilot_isa), "shots each:", PILOT_SHOTS)
print("First pilot circuit depth:", pilot_isa[0].depth())
display(pilot_isa[0].draw("mpl", idle_wires=False, fold=80))

# --- Notebook code cell 12 ---
pilot_sampler = Sampler(mode=backend)
pilot_sampler.options.max_execution_time = PILOT_JOB_CAP_S
pilot_job = pilot_sampler.run(pilot_isa, shots=PILOT_SHOTS)
PILOT_JOB_ID = pilot_job.job_id()
print("PILOT JOB ID — SAVE THIS NOW:", PILOT_JOB_ID)
pilot_result = pilot_job.result()
print("Pilot complete.")

# --- Notebook code cell 14 ---
def extract_pub_shots(pub_result, meta):
    sense_strings = pub_result.data.sense.get_bitstrings()
    readout_strings = pub_result.data.readout.get_bitstrings()
    if len(sense_strings) != len(readout_strings):
        raise RuntimeError("Sense/readout shot count mismatch")
    rows = []
    for shot_index, (sense_bits, readout_bits) in enumerate(zip(sense_strings, readout_strings)):
```

Appendix D. IBM 10-minute Colab code — listing 6 of 10

```
0211          rows.append({**meta, "shot_index": shot_index,
0212                       "transition_code": int(sense_bits.replace(" ", ""), 2),
0213                       "data_bit": int(readout_bits.replace(" ", ""), 2)})
0214      return rows
0215
0216  pilot_rows = []
0217  for pub, meta in zip(pilot_result, pilot_manifest):
0218      pilot_rows.extend(extract_pub_shots(pub, meta))
0219  pilot_df = pd.DataFrame(pilot_rows)
0220  pilot_df["split"] = np.where(pilot_df.shot_index < int(0.60*PILOT_SHOTS), "fit", "validation")
0221  pilot_df.to_csv("END_QM_pilot_raw.csv", index=False)
0222  print("Pilot shot records:", len(pilot_df))
0223
0224  def conditional_vectors(frame, split):
0225      subset = frame[frame.split == split]
0226      table = {}
0227      for (wi, ti, code_value), group in subset.groupby(["window_index", "duration_index",
"transition_code"]):
0228          item = {"n_x": 0, "n_y": 0, "mx": np.nan, "my": np.nan}
0229          for axis in ("x", "y"):
0230              axis_group = group[group.axis == axis]
0231              if len(axis_group):
0232                  item[f"n_{axis}"] = len(axis_group)
0233                  item[f"m{axis}"] = float(np.mean(1 - 2*axis_group.data_bit.to_numpy()))
0234          table[(int(wi), int(ti), int(code_value))] = item
0235      return table
0236
0237  fit_vectors = conditional_vectors(pilot_df, "fit")
0238  validation_vectors = conditional_vectors(pilot_df, "validation")
0239  correction_maps = {}
0240  lexicon_rows = []
0241
0242  for wi in range(len(WINDOWS)):
0243      for ti in range(len(durations_us)):
0244          cmap = {}
0245          for raw_code in range(64):
0246              item = fit_vectors.get((wi, ti, raw_code), {})
0247              nx, ny = item.get("n_x", 0), item.get("n_y", 0)
0248              mx, my = item.get("mx", np.nan), item.get("my", np.nan)
0249              supported = min(nx, ny) >= MIN_CODE_SUPPORT and np.isfinite(mx) and np.isfinite(my)
0250              fallback = code_is_ambiguous(raw_code) or not supported or ti == 0
0251              angle = 0.0 if fallback else float(np.clip(-math.atan2(my, mx), -MAX_CORRECTION_RAD,
MAX_CORRECTION_RAD))
0252              cmap[raw_code] = angle
```

Appendix D. IBM 10-minute Colab code — listing 7 of 10

```
0253              lexicon_rows.append({"window_index": wi, "duration_index": ti,
0254                                   "transition_code": raw_code, "n_x": nx, "n_y": ny,
0255                                   "mx": mx, "my": my, "correction_rad_pre_gate": angle,
0256                                   "fallback": fallback})
0257
0258          def aggregate_coherence(vectors, correction):
0259              total = 0.0 + 0.0j; weight = 0.0
0260              for raw_code in range(64):
0261                  item = vectors.get((wi, ti, raw_code), {})
0262                  if not (np.isfinite(item.get("mx", np.nan)) and np.isfinite(item.get("my",
np.nan))):
0263                      continue
0264                  n = min(item.get("n_x", 0), item.get("n_y", 0))
0265                  total += n * complex(item["mx"], item["my"]) * np.exp(1j*correction.get(raw_code,
0.0))
0266                  weight += n
0267              return abs(total / weight) if weight else 0.0
0268
0269          base = aggregate_coherence(validation_vectors, {})
0270          corrected = aggregate_coherence(validation_vectors, cmap)
0271          validation_gain = corrected / max(base, 1e-12)
0272          gate_pass = bool(ti > 0 and validation_gain > 1.0)
0273          if not gate_pass:
0274              cmap = {raw_code: 0.0 for raw_code in range(64)}
0275          correction_maps[(wi, ti)] = cmap
0276          for row in lexicon_rows:
0277              if row["window_index"] == wi and row["duration_index"] == ti:
0278                  row["pilot_validation_gain"] = validation_gain
0279                  row["pilot_gate_pass"] = gate_pass
0280                  row["correction_rad"] = cmap[row["transition_code"]]
0281
0282  lexicon_df = pd.DataFrame(lexicon_rows)
0283  lexicon_df.to_csv("END_QM_frozen_lexicon.csv", index=False)
0284  print("Enabled maps:", int(lexicon_df.groupby(["window_index",
"duration_index"]).pilot_gate_pass.first().sum()),
0285        "of", len(WINDOWS)*len(durations_us))
0286  display(lexicon_df.groupby(["window_index", "duration_index"])[["pilot_validation_gain",
"pilot_gate_pass"]].first())
0287
0288  # --- Notebook code cell 16 ---
0289  confirm_items = []
0290  for wi, window in enumerate(WINDOWS):
0291      for ti, duration in enumerate(durations_us):
0292          for axis in ("x", "y"):
0293              for method in METHODS:
0294                  if method == "end_qm":
```

Appendix D. IBM 10-minute Colab code — listing 8 of 10

```
                    qc = build_dynamic_circuit(window, duration, axis, correction_maps[(wi, ti)],
+1.0)
                elif method == "wrong_sign":
                    qc = build_dynamic_circuit(window, duration, axis, correction_maps[(wi, ti)],
-1.0)
                elif method == "matched_open":
                    qc = build_dynamic_circuit(window, duration, axis, None)
                else:
                    qc = build_reference_circuit(window, duration, axis, method)
                confirm_items.append((manifest_key(wi, ti, axis, method),
                                      {"key": manifest_key(wi, ti, axis, method),
                                       "window_index": wi, "duration_index": ti,
                                       "window_rad": float(window), "duration_us":
float(duration),
                                       "axis": axis, "method": method}, qc))

random.Random(SEED).shuffle(confirm_items)
assert len(confirm_items) == 480
confirm_manifest = [item[1] for item in confirm_items]
confirm_isa = pass_manager.run([item[2] for item in confirm_items])
chunks = [(confirm_isa[i:i+240], confirm_manifest[i:i+240]) for i in range(0, 480, 240)]
print("Confirmation circuits:", len(confirm_isa), "in chunks:", [len(c[0]) for c in chunks])
print("Total circuit executions planned:", len(pilot_isa)*PILOT_SHOTS +
len(confirm_isa)*CONFIRM_SHOTS)

# --- Notebook code cell 17 ---
CONFIRM_JOB_IDS = []
confirm_results = []
for chunk_index, (circuits, manifest) in enumerate(chunks):
    sampler = Sampler(mode=backend)
    sampler.options.max_execution_time = CONFIRM_JOB_CAP_S
    job = sampler.run(circuits, shots=CONFIRM_SHOTS)
    job_id = job.job_id()
    CONFIRM_JOB_IDS.append(job_id)
    print(f"CONFIRM JOB {chunk_index+1} ID — SAVE THIS NOW:", job_id)
    confirm_results.append((job.result(), manifest))
print("All confirmation jobs complete.")

# --- Notebook code cell 19 ---
confirm_rows = []
for chunk_result, manifest in confirm_results:
    for pub, meta in zip(chunk_result, manifest):
        readout_strings = pub.data.readout.get_bitstrings()
        # Raw-idle and XY4 controls intentionally leave the sentinel register
        # unmeasured. Some Runtime result versions omit such empty registers.
        try:
```

Appendix D. IBM 10-minute Colab code — listing 9 of 10

```
            sense_strings = pub.data.sense.get_bitstrings()
        except (AttributeError, KeyError):
            sense_strings = ["000000"] * len(readout_strings)
        for shot_index, readout_bits in enumerate(readout_strings):
            sense_bits = sense_strings[shot_index] if shot_index < len(sense_strings) else
"000000"
            confirm_rows.append({**meta, "shot_index": shot_index,
                                 "transition_code": int(sense_bits.replace(" ", ""), 2),
                                 "data_bit": int(readout_bits.replace(" ", ""), 2)})
confirm_df = pd.DataFrame(confirm_rows)
confirm_df.to_csv("END_QM_confirmation_raw.csv", index=False)

means = (confirm_df.assign(z=1-2*confirm_df.data_bit)
         .groupby(["window_index", "duration_index", "duration_us", "method", "axis"])
         .z.agg(["mean", "count"]).reset_index())
pivot = means.pivot_table(index=["window_index", "duration_index", "duration_us", "method"],
                          columns="axis", values="mean").reset_index()
pivot["coherence"] = np.hypot(pivot["x"], pivot["y"])

auc_rows = []
for (wi, method), group in pivot.groupby(["window_index", "method"]):
    group = group.sort_values("duration_us")
    normalized = group.coherence.to_numpy() / max(group.coherence.iloc[0], 1e-12)
    auc_rows.append({"window_index": int(wi), "method": method,
                     "auc_us": float(np.trapezoid(normalized, group.duration_us.to_numpy()))})
auc = pd.DataFrame(auc_rows)
auc_pivot = auc.pivot(index="window_index", columns="method", values="auc_us")
window_gain = (auc_pivot.end_qm / auc_pivot.matched_open).rename("gain").reset_index()

rng = np.random.default_rng(SEED)
boot = []
values = window_gain.gain.to_numpy()
for _ in range(20000):
    boot.append(float(np.median(rng.choice(values, size=len(values), replace=True))))
median_gain = float(np.median(values))
ci_low, ci_high = np.quantile(boot, [0.025, 0.975])
wrong_ratio = float(np.median(auc_pivot.end_qm / auc_pivot.wrong_sign))
screen_pass = bool(median_gain >= 1.10 and ci_low > 1.00 and wrong_ratio > 1.00)

decision = {
    "status": "PASS" if screen_pass else "DO_NOT_CLAIM_PASS",
    "backend": backend.name,
    "physical_qubits": physical_qubits,
```

Appendix D. IBM 10-minute Colab code — listing 10 of 10

```
0379      "pilot_job_id": PILOT_JOB_ID,
0380      "confirmation_job_ids": CONFIRM_JOB_IDS,
0381      "median_auc_gain_vs_matched": median_gain,
0382      "paired_window_bootstrap_ci95": [float(ci_low), float(ci_high)],
0383      "median_end_vs_wrong_sign": wrong_ratio,
0384      "screening_threshold": "median>=1.10; lower95>1.00; END>wrong-sign",
0385      "claim_boundary": "One-device hardware screen only; repeat on >=3 dates and >=2 data qubits
before a general hardware claim."
0386  }
0387  Path("END_QM_hardware_decision.json").write_text(json.dumps(decision, indent=2))
0388  auc.to_csv("END_QM_hardware_auc.csv", index=False)
0389  print(json.dumps(decision, indent=2))
0390
0391  fig, axes = plt.subplots(1, 2, figsize=(12, 4.3))
0392  for method, group in pivot.groupby("method"):
0393      curve = group.groupby("duration_us").coherence.median()
0394      axes[0].plot(curve.index, curve.values, marker="o", label=method)
0395  axes[0].set(xlabel="Delay (us)", ylabel="Median equatorial coherence", title="Held-out hardware
curves")
0396  axes[0].legend(fontsize=8)
0397  axes[1].bar(window_gain.window_index, window_gain.gain, color="#2E74B5")
0398  axes[1].axhline(1.0, color="black", lw=1); axes[1].axhline(1.10, color="#C45A35", ls="--")
0399  axes[1].set(xlabel="Start-phase window", ylabel="END-QM / matched AUC", title="All-window gain")
0400  fig.tight_layout(); fig.savefig("END_QM_hardware_results.png", dpi=220)
0401  plt.show()
0402
0403  # --- Notebook code cell 21 ---
0404  from google.colab import files
0405  for filename in [
0406      "END_QM_pilot_raw.csv", "END_QM_frozen_lexicon.csv",
0407      "END_QM_confirmation_raw.csv", "END_QM_hardware_auc.csv",
0408      "END_QM_hardware_decision.json", "END_QM_hardware_results.png",
0409  ]:
0410      if Path(filename).exists():
0411          files.download(filename)
0412
```