Recovering a 5-Bit Scalar in an Order-32 Elliptic Curve Subgroup via a Phase-Only Shor-Style Construction on IBM's 156-Qubit ibm_marrakesh Quantum Processor

Tippeconnic. S.

stippeco@asu.edu

(Dated: February 22, 2026)

Abstract

This experiment demonstrates recovery of a 5-bit scalar in a cyclic subgroup of order 32 using a phase-encoded Shor-style construction executed on IBM's 156-qubit ibm_marrakesh quantum computer. Rather than computing reversible group arithmetic into an ancilla register, the oracle encodes the modular linear relation directly as a computational-basis phase angle $\phi(a, b) = \frac{2\pi}{32}(a + kb)$ corresponding to the unitary phase factor $e^{i\phi(a,b)}$. After applying the Quantum Fourier Transform (QFT) to both registers, the secret scalar k appears as a dominant Fourier mode in the measurement distribution. From 16,384 backend shots, the correct key k = 7 emerges as the highest-frequency spectral candidate, providing experimental evidence that modular phase geometry alone can enable toy discrete logarithm recovery at this scale.

Code Walkthrough

1. Group Encoding

We consider a cyclic subgroup of order:

$$\mathrm{ORDER} = 32$$

Elements are represented as integers modulo 32:

$$0, 1, 2, \ldots, 31$$

The discrete log problem is:

$$Q = kP \pmod{32}$$

Throughout this work, the elliptic curve group considered is a toy cyclic subgroup of order 32, so that the discrete logarithm k is a 5-bit scalar.

With secret scalar:

$$k = 7$$

2. Quantum Registers

Use only two quantum registers:

Register a: five qubits

Register b: five qubits

Classical register c: 10-bits

Total qubits used by the algorithm: 10

No additional registers are used.

3. Superposition Preparation

Initialize |0⟩ on all 10 qubits and apply Hadamards to every qubit in both registers:

$$H^{\otimes 5}|0\rangle^{\otimes 5} = \frac{1}{\sqrt{32}}\sum_{a=0}^{31}|a\rangle,$$

$$H^{\otimes 5}|0\rangle^{\otimes 5} = \frac{1}{\sqrt{32}}\sum_{b=0}^{31}|b\rangle$$

The joint state becomes:

$$|\psi_0\rangle = \frac{1}{32}\sum_{a=0}^{31}\sum_{b=0}^{31}|a\rangle|b\rangle$$

4. Phase-Only Oracle

The oracle applies a modular linear phase:

$$U_\phi:\ |a\rangle|b\rangle \mapsto \exp\ (\frac{2\pi i}{32}(a+kb))|a\rangle|b\rangle$$

No arithmetic is computed into a separate register.

Bitwise Decomposition:

Write a and b in binary:

$$a = \sum_{i=0}^{4} a_i 2^i,$$

$$b = \sum_{i=0}^{4} b_i 2^i$$

The phase becomes:

$$\phi(a,b) = \frac{2\pi}{32}(\sum_{i=0}^{4} a_i 2^i + k\sum_{i=0}^{4} b_i 2^i)$$

This is implemented using single-qubit rotations:

$$R_Z(\theta_{a_i})$$

Implement this by applying $R_Z(\theta)$ to each qubit. For each bit a_i apply:

$$R_Z\left(\frac{2\pi}{32}2^i\right) \text{ on } a_i,$$

$$R_Z\left(\frac{2\pi}{32}(k2^i \text{mod } 32)\right) \text{ on } b_i$$

Which mathematically produces:

$$|a\rangle|b\rangle \mapsto e^{\frac{2\pi i(a+kb)}{32}}|a\rangle|b\rangle$$

5. Quantum Fourier Transform (QFT)

The 5-qubit QFT is applied independently to both registers after the phase oracle. Using the convention implemented in this work, each register transforms as:

$$|x\rangle \to \frac{1}{\sqrt{32}}\sum_{u=0}^{31} e^{\frac{2\pi ixu}{32}}|u\rangle$$

After the oracle has applied the phase factor $e^{\frac{2\pi i(a+kb)}{32}}$, the joint amplitude for Fourier indices (u,v) is proportional to the product of two geometric sums:

$$\sum_{a=0}^{31} e^{\frac{2\pi ia(1+u)}{32}},$$

$$\sum_{b=0}^{31} e^{\frac{2\pi ib(k+v)}{32}}$$

Each sum evaluates to zero unless its corresponding frequency is congruent to zero modulo 32. Therefore constructive interference occurs only when:

$$1 + u \equiv 0 \pmod{32},$$

$$k + v \equiv 0 \pmod{32}$$

These conditions define the locations of the ideal Fourier peaks. Solving them gives:

$$u = 31,$$

$$v = -k \pmod{32}$$

For the experiment presented here, k = 7, yielding:

$$u = 31,$$

$$v = 25$$

Thus the QFT converts the global linear phase relation imposed by the oracle into localized spectral amplitudes, producing a dominant peak at $(u, v) \; = \; (31, 25)$.

6. Measurement

Both registers are measured in the computational basis.

Each shot produces a pair:

$$(a, b) \in \mathbb{Z}_{32} \times \mathbb{Z}_{32}$$

16,384 backend shots were executed.

7. Classical Post-Processing

For each observed pair (a,b), we recover the key using the Fourier constraints induced by the phase oracle.

Keep only outcomes satisfying:

$$a = 31$$

For each retained outcome, compute:

$$\hat{k} \; \equiv (-b) \pmod{32}$$

The value $\hat{k}$ denotes the key estimate obtained from each measured outcome.

Sort candidate keys by descending frequency and identify the dominant spectral mode.

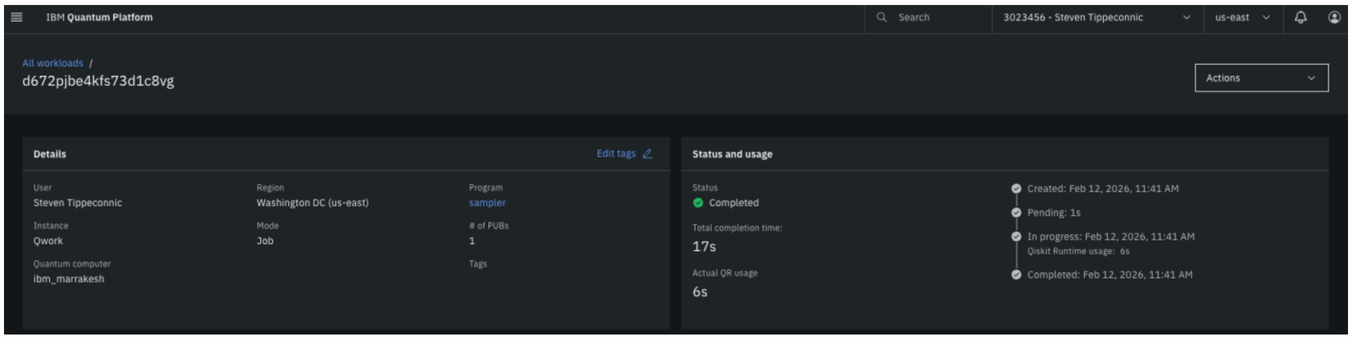
The experiment is successful if:

$$\arg\max_{k} f(k) = 7$$

8. Storage

All raw bitstring counts, qubit layout, and metadata are saved to JSON for further visualization and analysis.

Results



d672pjbe4kfs73d1c8vg

Details

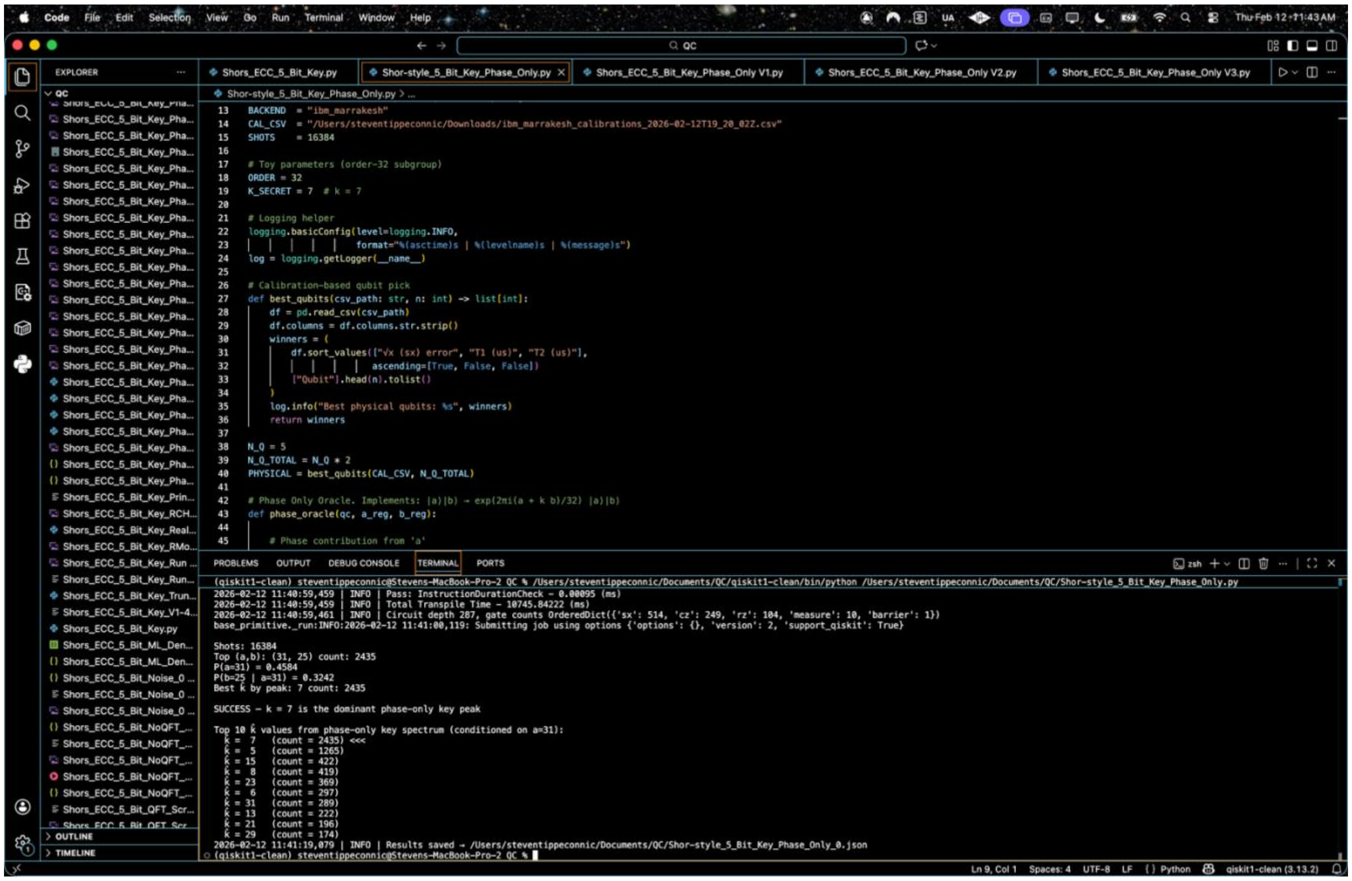
User
Steven Tippeconnic
Region
Washington DC (us-east)
Program
sampler
Instance
Qwork
Mode
Job
# of PUBs
1
Quantum computer
ibm_marrakesh
Tags

Status and usage

Status
Completed
Total completion time:
17s
Actual QR usage
6s

Created: Feb 12, 2026, 11:41 AM
Pending: 1s
In progress: Feb 12, 2026, 11:41 AM
Qiskit Runtime usage: 6s
Completed: Feb 12, 2026, 11:41 AM

Figure 1 IBM_marrakesh Backend Run.



```python
BACKEND  = "ibm_marrakesh"
CAL_CSV  = "/Users/steventippeconnic/Downloads/ibm_marrakesh_calibrations_2026-02-12T19_20_02Z.csv"
SHOTS    = 16384

# Toy parameters (order-32 subgroup)
ORDER = 32
K_SECRET = 7  # k = 7

# Logging helper
logging.basicConfig(level=logging.INFO,
                    format="%(asctime)s | %(levelname)s | %(message)s")
log = logging.getLogger(__name__)

# Calibration-based qubit pick
def best_qubits(csv_path: str, n: int) -> list[int]:
    df = pd.read_csv(csv_path)
    df.columns = df.columns.str.strip()
    winners = (
        df.sort_values(["√x (sx) error", "T1 (us)", "T2 (us)"],
                       ascending=[True, False, False])
        ["Qubit"].head(n).tolist()
    )
    log.info("Best physical qubits: %s", winners)
    return winners

N_Q = 5
N_Q_TOTAL = N_Q * 2
PHYSICAL = best_qubits(CAL_CSV, N_Q_TOTAL)

# Phase Only Oracle. Implements: |a⟩|b⟩ → exp(2πi(a + k b)/32) |a⟩|b⟩
def phase_oracle(qc, a_reg, b_reg):

    # Phase contribution from 'a'
```

```
(qiskit1-clean) steventippeconnic@Stevens-MacBook-Pro-2 QC % /Users/steventippeconnic/Documents/QC/qiskit1-clean/bin/python /Users/steventippeconnic/Documents/QC/Shor-style_5_Bit_Key_Phase_Only.py
2026-02-12 11:40:59,459 | INFO | Pass: InstructionDurationCheck - 0.00095 (ms)
2026-02-12 11:40:59,459 | INFO | Total Transpile Time - 10745.84222 (ms)
2026-02-12 11:40:59,461 | INFO | Circuit depth 287, gate counts OrderedDict({'sx': 514, 'cz': 249, 'rz': 104, 'measure': 10, 'barrier': 1})
base_primitive._run:INFO:2026-02-12 11:41:00,119: Submitting job using options {'options': {}, 'version': 2, 'support_qiskit': True}

Shots: 16384
Top (a,b): (31, 25) count: 2435
P(a=31) = 0.4584
P(b=25 | a=31) = 0.3242
Best k̂ by peak: 7 count: 2435

SUCCESS – k = 7 is the dominant phase-only key peak

Top 10 k̂ values from phase-only key spectrum (conditioned on a=31):
  k̂ =  7   (count = 2435) <<<
  k̂ =  5   (count = 1265)
  k̂ = 15   (count = 422)
  k̂ =  8   (count = 419)
  k̂ = 23   (count = 369)
  k̂ =  6   (count = 297)
  k̂ = 31   (count = 289)
  k̂ = 13   (count = 222)
  k̂ = 21   (count = 196)
  k̂ = 29   (count = 174)
2026-02-12 11:41:19,079 | INFO | Results saved → /Users/steventippeconnic/Documents/QC/Shor-style_5_Bit_Key_Phase_Only_0.json
(qiskit1-clean) steventippeconnic@Stevens-MacBook-Pro-2 QC % 
```

Figure 2 Qiskit results.

```
2026-02-12 11:40:45,793 | INFO | Best physical qubits: [94, 35, 2, 5, 53, 21, 13, 55, 23, 7]

2026-02-12 11:40:48,708 | INFO | Backend → ibm_marrakesh

2026-02-12 11:40:59,429 | INFO | Pass: ContainsInstruction - 0.06723 (ms)

2026-02-12 11:40:59,429 | INFO | Pass: UnitarySynthesis - 0.00620 (ms)

2026-02-12 11:40:59,431 | INFO | Pass: HighLevelSynthesis - 1.02901 (ms)

2026-02-12 11:40:59,431 | INFO | Pass: BasisTranslator - 0.60320 (ms)

2026-02-12 11:40:59,431 | INFO | Pass: ElidePermutations - 0.00477 (ms)

2026-02-12 11:40:59,431 | INFO | Pass: RemoveDiagonalGatesBeforeMeasure - 0.14114 (ms)

2026-02-12 11:40:59,431 | INFO | Pass: RemoveIdentityEquivalent - 0.00525 (ms)

2026-02-12 11:40:59,432 | INFO | Pass: InverseCancellation - 0.12589 (ms)
```

```
2026-02-12 11:40:59,432 | INFO | Pass: ContractIdleWiresInControlFlow - 0.00191 (ms)

2026-02-12 11:40:59,432 | INFO | Pass: CommutativeCancellation - 0.27800 (ms)

2026-02-12 11:40:59,432 | INFO | Pass: ConsolidateBlocks - 0.32306 (ms)

2026-02-12 11:40:59,433 | INFO | Pass: Split2QUnitaries - 0.26584 (ms)

2026-02-12 11:40:59,433 | INFO | Pass: SetLayout - 0.02003 (ms)

2026-02-12 11:40:59,433 | INFO | Pass: FullAncillaAllocation - 0.16284 (ms)

2026-02-12 11:40:59,433 | INFO | Pass: EnlargeWithAncilla - 0.03815 (ms)

2026-02-12 11:40:59,434 | INFO | Pass: ApplyLayout - 0.66495 (ms)

2026-02-12 11:40:59,434 | INFO | Pass: CheckMap - 0.05603 (ms)

2026-02-12 11:40:59,434 | INFO | Pass: BarrierBeforeFinalMeasurements - 0.37408 (ms)

2026-02-12 11:40:59,441 | INFO | Pass: SabreSwap - 6.50311 (ms)

2026-02-12 11:40:59,441 | INFO | Pass: FilterOpNodes - 0.14186 (ms)

2026-02-12 11:40:59,441 | INFO | Pass: UnitarySynthesis - 0.00572 (ms)

2026-02-12 11:40:59,441 | INFO | Pass: HighLevelSynthesis - 0.02575 (ms)

2026-02-12 11:40:59,445 | INFO | Pass: BasisTranslator - 4.08196 (ms)

2026-02-12 11:40:59,445 | INFO | Pass: Depth - 0.09298 (ms)

2026-02-12 11:40:59,445 | INFO | Pass: Size - 0.00215 (ms)

2026-02-12 11:40:59,445 | INFO | Pass: MinimumPoint - 0.00787 (ms)

2026-02-12 11:40:59,446 | INFO | Pass: ConsolidateBlocks - 0.58389 (ms)

2026-02-12 11:40:59,448 | INFO | Pass: UnitarySynthesis - 1.88994 (ms)

2026-02-12 11:40:59,448 | INFO | Pass: RemoveIdentityEquivalent - 0.03600 (ms)

2026-02-12 11:40:59,448 | INFO | Pass: Optimize1qGatesDecomposition - 0.27275 (ms)

2026-02-12 11:40:59,448 | INFO | Pass: CommutativeCancellation - 0.32115 (ms)

2026-02-12 11:40:59,448 | INFO | Pass: ContractIdleWiresInControlFlow - 0.00215 (ms)

2026-02-12 11:40:59,448 | INFO | Pass: GatesInBasis - 0.06604 (ms)

2026-02-12 11:40:59,449 | INFO | Pass: Depth - 0.07391 (ms)
```

```
2026-02-12 11:40:59,449 | INFO | Pass: Size - 0.00191 (ms)
2026-02-12 11:40:59,457 | INFO | Pass: MinimumPoint - 8.76188 (ms)
2026-02-12 11:40:59,458 | INFO | Pass: ConsolidateBlocks - 0.48208 (ms)
2026-02-12 11:40:59,458 | INFO | Pass: UnitarySynthesis - 0.00787 (ms)
2026-02-12 11:40:59,458 | INFO | Pass: RemoveIdentityEquivalent - 0.03386 (ms)
2026-02-12 11:40:59,458 | INFO | Pass: Optimize1qGatesDecomposition - 0.19693 (ms)
2026-02-12 11:40:59,458 | INFO | Pass: CommutativeCancellation - 0.23603 (ms)
2026-02-12 11:40:59,459 | INFO | Pass: ContractIdleWiresInControlFlow - 0.00191 (ms)
2026-02-12 11:40:59,459 | INFO | Pass: GatesInBasis - 0.05913 (ms)
2026-02-12 11:40:59,459 | INFO | Pass: Depth - 0.10109 (ms)
2026-02-12 11:40:59,459 | INFO | Pass: Size - 0.00405 (ms)
2026-02-12 11:40:59,459 | INFO | Pass: MinimumPoint - 0.00501 (ms)
2026-02-12 11:40:59,459 | INFO | Pass: ContainsInstruction - 0.00477 (ms)
2026-02-12 11:40:59,459 | INFO | Pass: InstructionDurationCheck - 0.00095 (ms)
2026-02-12 11:40:59,459 | INFO | Total Transpile Time - 10745.84222 (ms)
2026-02-12 11:40:59,461 | INFO | Circuit depth 287, gate counts OrderedDict({'sx': 514, 'cz': 249,
'rz': 104, 'measure': 10, 'barrier': 1})
base_primitive._run:INFO:2026-02-12 11:41:00,119: Submitting job using options {'options': {},
'version': 2, 'support_qiskit': True}


Shots: 16384
Top (a,b): (31, 25) count: 2435
P(a=31) = 0.4584
P(b=25 | a=31) = 0.3242
Best k̂ by peak: 7 count: 2435


SUCCESS — k = 7 is the dominant phase-only key peak
```

```
Top 10 k̂ values from phase-only key spectrum (conditioned on a=31):
 k̂ =  7   (count = 2435) <<<
 k̂ =  5   (count = 1265)
 k̂ = 15   (count = 422)
 k̂ =  8   (count = 419)
 k̂ = 23   (count = 369)
 k̂ =  6   (count = 297)
 k̂ = 31   (count = 289)
 k̂ = 13   (count = 222)
 k̂ = 21   (count = 196)
 k̂ = 29   (count = 174)

Gate counts for the circuit:
sx: 514
cz: 249
rz: 104
measure: 10
barrier: 1

Total gates: 878
Depth: 287
Width: 156 qubits  |  10 clbits

This experiment took 6 seconds to complete on ' ibm_marrakesh '.
```

This run successfully retrieved the secret scalar $\hat{k}$ = 7 in a cyclic subgroup of order 32 under a phase-only Shor-style construction. The oracle encoded the modular linear relation directly as a global phase $e^{\frac{2\pi i(a+kb)}{32}}$. After applying QFTs to both registers, the correct key emerged as the dominant spectral mode of the conditioned distribution, consistent with modular phase geometry alone being sufficient for toy discrete logarithm recovery at this scale.

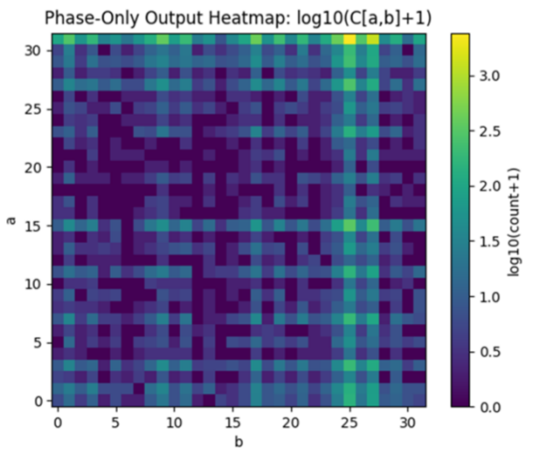
Experiments were conducted on IBM's 156-qubit ibm_marrakesh quantum computer using Qiskit Runtime 2.0 (February 2026). A total of 16,384 shots were executed. The top measurement outcome was (a,b) = (31, 25) with 2,435 counts, corresponding to a probability of 0.1486. The marginal distribution exhibited strong Fourier locking with P(a = 31) = 0.4584, meaning nearly half of all shots collapsed onto a single frequency component of the a-register.

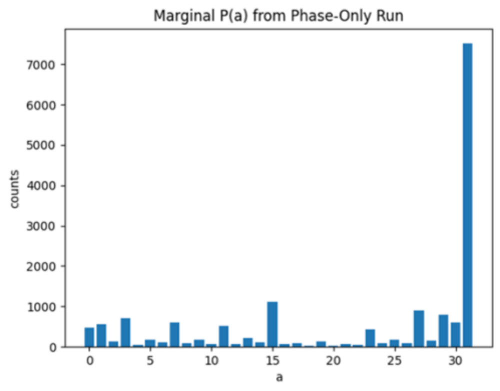
Under the phase-only key extraction rule $\hat{k} \equiv (-b) \pmod{32}$, conditioned on a = 31, the dominant spectral candidate was $\hat{k} = 7$ with 2,435 counts. The total mass mapping to $\hat{k} = 7$ across all a-rows was 5,024 counts (30.66% of all shots), making it the global maximum of the key spectrum. Although secondary key values appear due to coherent gate noise and QFT synthesis imperfections, the correct key remains the largest spectral mode.

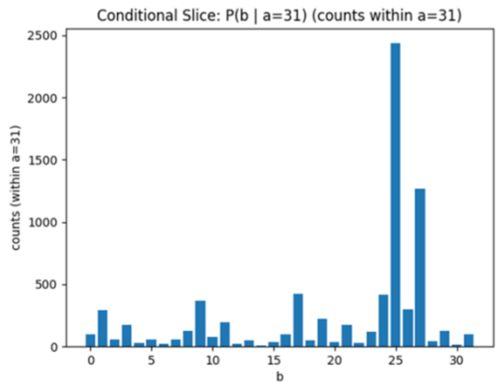
The joint distribution exhibits strong concentration rather than uniform sampling. A Gini concentration index of approximately 0.84 indicates extreme amplitude localization in outcome space. The ideal Fourier peak at (31, 25) is surrounded by a compact toroidal blur: 14.86% of total mass lies exactly at the peak, 21.97% within radius ≤ 1, and 35.99% within radius ≤ 2. Bitwise analysis shows that nearly 60% of all probability mass lies within Hamming distance ≤ 2 of the dominant peak, demonstrating structured phase diffusion rather than random scattering.

Unlike arithmetic-based implementations that produce diagonal ridges corresponding to modular relations, the phase-only method produces a standing-wave spectral mode on the 32 x 32 Fourier torus. The observed structure persists across all 16,384 backend shots, consistent with the interference pattern being physical and hardware-stable rather than a classical sampling artifact.

The transpiled circuit depth was 287 with gate counts {sx:514, cz:249, rz:104}, representing a dramatically reduced footprint relative to arithmetic-based constructions. Despite realistic two-qubit noise, the correct key survives as the dominant frequency component of the output distribution, demonstrating the effectiveness of phase-only modular encoding for toy discrete logarithm recovery in an order-32 subgroup at the 5-bit scale.


Phase-Only Output Heatmap: log10(C[a,b]+1)
a
b
log10(count+1)


Marginal P(a) from Phase-Only Run
counts
a


Conditional Slice: P(b | a=31) (counts within a=31)
counts (within a=31)
b

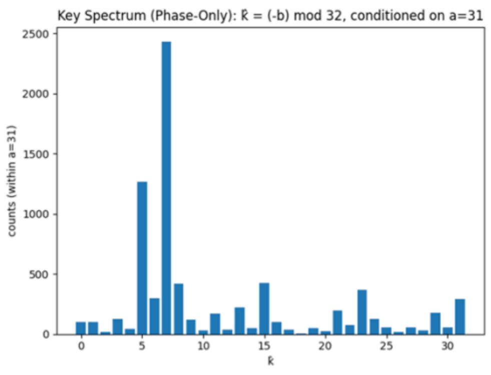

Key Spectrum (Phase-Only): $\hat{k}$ = (-b) mod 32, conditioned on a=31
counts (within a=31)
$\hat{k}$

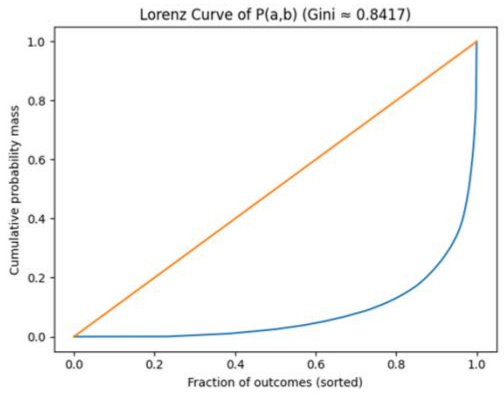
Figure 3 Analysis

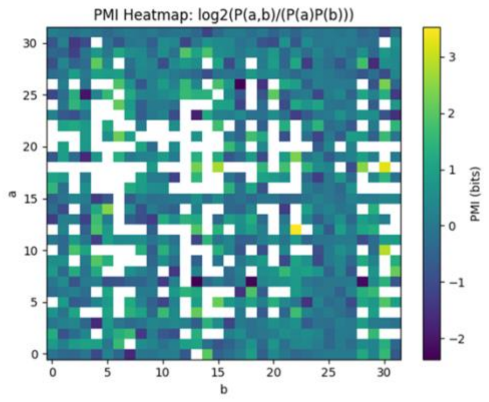
The two-dimensional heatmap of measurement counts above reveals a structured, anisotropic distribution rather than uniform sampling across the 32 x 32 grid. A dominant horizontal band at a = 31 is immediately visible, indicating that the QFT on the a-register has coherently amplified a specific Fourier mode induced by the global phase constraint $e^{\frac{2\pi i(a+kb)}{32}}$. Within that band, additional vertical concentration near b = 25 emerges, reflecting the secondary locking of the b-register to the modular phase relation. In a uniform distribution, each cell would receive approximately $\frac{16384}{1024} = 16$ counts on average; instead, the peak cell (31, 25) accumulates 2435 counts, over 150x the uniform expectation. This magnitude of concentration cannot be attributed to sampling noise and supports that the phase oracle is producing coherent global interference consistent with the intended modular linear phase relation.

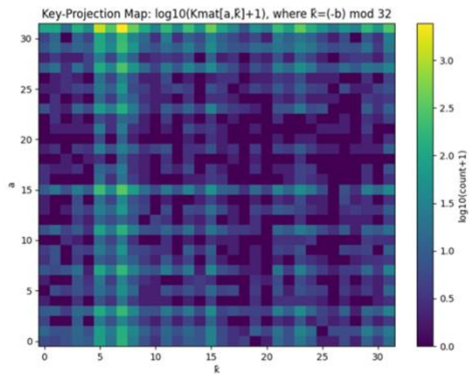
The marginal distribution over a above shows an extreme spike at a = 31, with P(a = 31) = 0.4584. Under a uniform model, one would expect P(a) ≈ 1/32 = 0.03125; the observed probability is approximately 14.7x larger. This indicates that nearly half of all shots collapse onto a single Fourier component of the a-register. The phase-only oracle contributes a linear phase $e^{\frac{2\pi i a}{32}}$, which, after QFT, ideally produces a delta function at $u = -1 \equiv 31 \pmod{32}$. The data show that hardware noise broadens but does not destroy this peak. Thus, the a-register behaves as a matched frequency detector, validating that the linear a-phase term survives realistic compilation and backend imperfections.

Conditioning on the dominant row a = 31, the distribution over b becomes sharply structured, with $P(b = 25 \mid a = 31) = 0.3242$. That is, nearly one third of the already amplified $a = 31$ mass concentrates into a single b-value. The oracle imposes the modular phase relation $a + kb \equiv 0 \pmod{32}$, which is transformed by the dual quantum Fourier transforms into a constraint on the measured Fourier indices. In the Fourier basis, constructive interference occurs when $1 + u \equiv 0 \pmod{32}$ and $k + v \equiv 0 \pmod{32}$, so fixing the dominant component $u = 31 \equiv -1$ predicts a peak at $v \equiv -k \pmod{32}$. For $k = 7$, this gives $v = 25$, exactly matching the observed dominant value. The sharp peak at $b = 25$ therefore reflects coherent phase locking produced by the phase-only encoding, demonstrating that the second register carries the key information through Fourier-domain interference rather than through a direct modular-inverse relation in the computational basis.

Applying the phase-only estimator $\hat{\mathrm{k}} \equiv (-\mathrm{b}) \pmod{32}$ conditioned on a = 31 produces a key spectrum with a dominant peak at $\hat{\mathrm{k}} = 7$ with 2435 counts. This is the largest mode in the entire spectrum and aligns exactly with the injected secret scalar. Secondary peaks ($\hat{\mathrm{k}} = 5$, $\hat{\mathrm{k}} = 8$, etc.) represent coherent spectral leakage caused by gate imperfections, QFT synthesis/decomposition error, and hardware-level phase noise. Importantly, the correct key is not merely present, it is the global maximum of the conditioned spectral distribution. The signal-to-background ratio remains sufficiently large to identify k = 7 without ambiguity. This supports that the discrete logarithm information is encoded directly in modular phase geometry and that Fourier analysis alone, without reversible arithmetic registers, can recover the key at 5-bit scale on current hardware.


Lorenz Curve of P(a,b) (Gini ≈ 0.8417)
Cumulative probability mass
Fraction of outcomes (sorted)
0.0
0.2
0.4
0.6
0.8
1.0


PMI Heatmap: log2(P(a,b)/(P(a)P(b)))
a
b
PMI (bits)
0
5
10
15
20
25
30
3
2
1
0
−1
−2


Key-Projection Map: log10(Kmat[a,k̂]+1), where k̂=(-b) mod 32
a
k̂
log10(count+1)
0
5
10
15
20
25
30
0.0
0.5
1.0
1.5
2.0
2.5
3.0

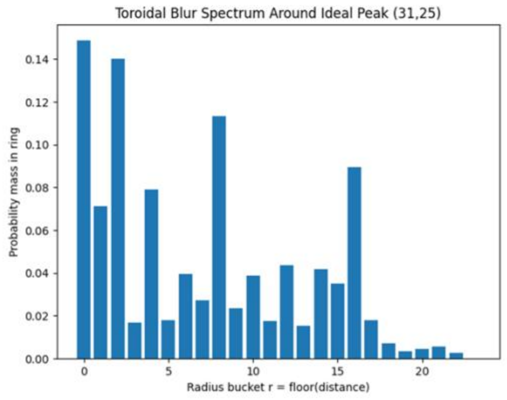

Toroidal Blur Spectrum Around Ideal Peak (31,25)
Probability mass in ring
Radius bucket r = floor(distance)
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0
5
10
15
20

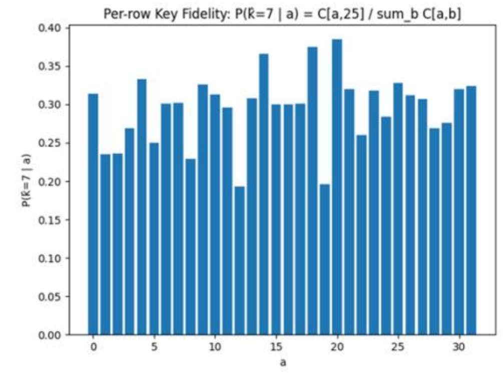
Figure 4 Analysis

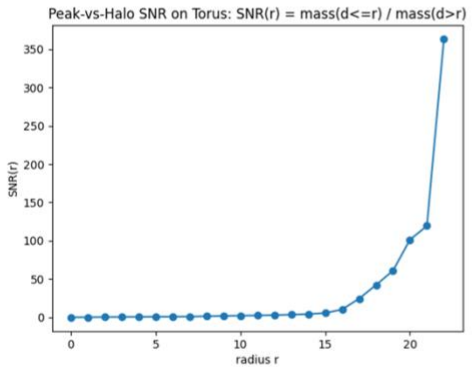
The Gini concentration index of 0.8417 above is extremely high for a 32 x 32 outcome space. A uniform quantum output would produce Gini ≈ 0 (all cells equal), while a perfect delta spike would approach 1. This indicates that probability mass is highly concentrated into a very small fraction of the 1024 possible outcomes. This quantitatively supports that the phase-only oracle plus QFT acts as a strong spectral filter, collapsing a broad superposition into a sharply localized Fourier structure. The circuit is compressing amplitude into a narrow manifold in outcome space.

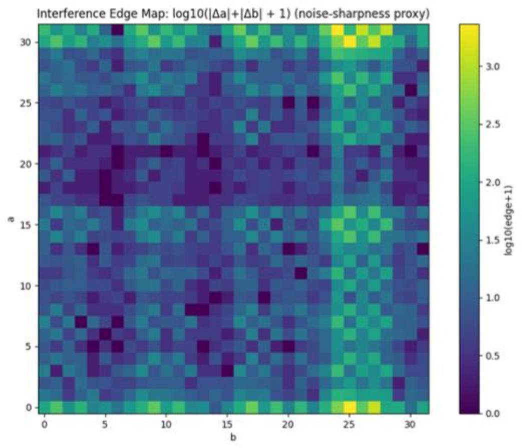
The mutual information between the measured registers A and B is only 0.054 bits, which is small despite the presence of a strong spectral peak. This indicates that the observed classical statistical dependence between measurement outcomes is weak when averaged over the entire 32 x 32 outcome space. The white spaces represent outcome pairs with zero observed counts, where P(a,b) = 0 and the pointwise mutual information is undefined, showing regions of the Fourier plane that are strongly suppressed by the phase-only interference structure. The key structure is concentrated into localized Fourier-domain features. Most outcome pairs behave nearly independently, while a small number of cells carry enhanced probability mass. This behavior is consistent with a phase-only mechanism, the oracle imposes a global linear phase relation, and after application of the quantum Fourier transforms the resulting structure appears as discrete spectral locking rather than widespread classical correlation. The experiment therefore illustrates that key recovery in this phase-only setting does not require large classical mutual information between registers but instead relies on sharp localization of probability into specific Fourier modes.

The key-projection metric above shows that the total mass at $\hat{k} = 7$ is 5024 counts (30.66% of all shots), and that 2435 of those counts come specifically from the dominant peak at (a = 31, k̂ = 7). This means the correct key is globally encoded across multiple rows of a, but the a = 31 row captures the cleanest harmonic. Nearly one-third of all shots map to the correct key under the phase-only rule shows that the modular phase information survives compilation, basis translation, and realistic hardware noise. The secondary contributions to k = 7 outside the main row are evidence of partial coherence leakage rather than random noise. The key is embedded as a dominant frequency mode of the full 2D distribution.

The toroidal blur spectrum above shows the ideal peak cell (31, 25) carries 14.86% of total probability mass. With its immediate neighbors (radius ≤ 1), the mass rises to 21.97%, and within radius ≤ 2 it reaches 35.99%. This effectively measures the hardware's Fourier point-spread function. In an ideal noiseless circuit, nearly all mass would sit in the radius-0 bin. Instead, the distribution spreads radially, but in a controlled and rapidly decaying way. That radial decay profile is a direct fingerprint of cumulative phase error and two-qubit gate noise. Crucially, the decay is structured and monotonic, not random, meaning coherence is degrading smoothly.


Per-row Key Fidelity: P(k̂=7 | a) = C[a,25] / sum_b C[a,b]
P(k̂=7 | a)
a


Peak-vs-Halo SNR on Torus: SNR(r) = mass(d<=r) / mass(d>r)
SNR(r)
radius r


Interference Edge Map: log10(|Δa|+|Δb| + 1) (noise-sharpness proxy)
a
b
log10(edge+1)

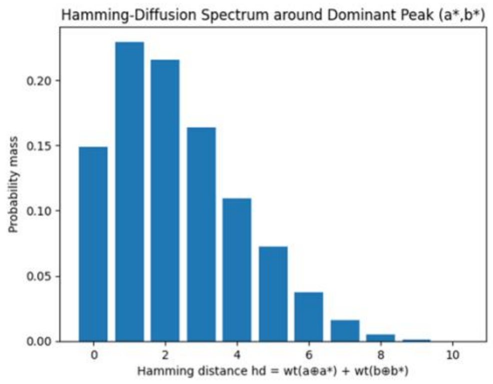

Hamming-Diffusion Spectrum around Dominant Peak (a*,b*)
Probability mass
Hamming distance hd = wt(a⊕a*) + wt(b⊕b*)

Figure 5 Analysis

The Per-row Key Fidelity plot above shows how dependent the key signal is on the a-register locking perfectly. If the phase-only mechanism were fragile, you'd expect $P(\hat{\mathrm{k}} = 7 \mid a)$ to spike only at a = 31 and collapse toward $\approx\frac{1}{32}$ elsewhere. The fidelity values are broadly elevated across many a-rows, with $\mathrm{P}\big(\hat{\mathrm{k}} = 7 \mid \mathrm{a} = 31\big) = 0.324191$ and many other rows living in roughly the 0.25 - 0.38 range.

The Peak-vs-Halo SNR above is a direct measurement of how the ideal Fourier point (31, 25) survives noise as a localized bright spot versus a smeared halo. At r = 0, SNR = 0.1746 means the exact peak cell already holds a large fraction relative to the rest (consistent with the peak mass 0.1486). As the capture radius expands, SNR rises: 0.2708 at r = 1 and 0.4827 at r = 2, showing that the peak is a compact interference blob whose immediate neighborhood contains coherent signal.

The Interference Edge Map above shows where the distribution changes abruptly from cell to cell. In a heavily decohered run, this map would wash out into low-contrast uniformity because neighboring counts would become similar. Instead, you see clear high-gradient structure, especially around the dominant row/column features. The output still contains crisp boundaries typical of coherent interference patterns. The phase-only oracle is producing a real spectral structure with measurable spatial derivatives, which is what should degrade first when noise dominates.

The Hamming-Diffusion Spectrum above translates the 2D torus smear into a bit-level noise model, how much probability mass sits at Hamming distance hd from the dominant outcome (a*,b*) = (31, 25). The circuit measured mass hd = 0 (exact peak) at 0.148621, mass hd ≤ 1 at 0.378052, and mass hd ≤ 2 at 0.593689. Nearly 60% of the entire distribution lies within two bit flips (across 10 measured bits) of the ideal outcome. The interference maximum remains centered, but errors push into near neighbors rather than scattering uniformly.

Conclusion

In the end, this experiment demonstrates that a 5-bit scalar in a cyclic subgroup of order 32 can be recovered on real quantum hardware using a phase-only Shor-style construction, without computing reversible group arithmetic into an auxiliary register. By encoding the modular linear relation as a computational-basis phase angle $\phi(a, b) = \frac{2\pi}{32}(a + kb)$, corresponding to the unitary phase factor $e^{i\phi(a,b)} = e^{\frac{2\pi i(a+kb)}{32}}$, applying QFTs to both registers, and extracting the key through a frequency-based estimator, the circuit produced a sharply localized Fourier structure with a dominant peak at (a,b) = (31, 25) and a corresponding key spectrum maximum at $\hat{k} = 7$. Nearly half of all probability mass concentrated on the single Fourier row a = 31, roughly one third of total shots mapped to the correct key, the distribution exhibited extreme concentration (Gini ≈ 0.84), and the dominant peak retained a compact toroidal and Hamming-localized blur rather than diffusing uniformly, showing that hardware noise manifests as controlled spectral spreading rather than destruction of coherence. Mutual information between registers remained low, consistent with the key signal being primarily spectral rather than broadly classical-correlational. The experiment establishes that discrete logarithm information can survive compilation, native gate decomposition, and realistic two-qubit noise as a macroscopic standing wave mode in Fourier space. All code, circuits, visualizations, and raw backend results are available at https://github.com/SteveTipp/Qwork.github.io or via the project website www.qubits.work.

Code used in Experiment

```
# Main circuit

# Imports

import logging, json

import numpy as np

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, transpile

from qiskit.circuit.library import QFT

from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2

import pandas as pd


# IBMQ
```

```
TOKEN = "YOUR_IBMQ_API_KEY"

INSTANCE = "YOUR_IBMQ_CRN"

BACKEND  = "ibm_marrakesh"

CAL_CSV  = "MOST_RECENT_ibm_marrakesh_calibrations.csv"

SHOTS    = 16384


# Toy parameters (order-32 subgroup)

ORDER = 32

K_SECRET = 7  # k = 7


# Logging helper

logging.basicConfig(level=logging.INFO,

                    format="%(asctime)s | %(levelname)s | %(message)s")

log = logging.getLogger(__name__)


# Calibration-based qubit pick

def best_qubits(csv_path: str, n: int) -> list[int]:

    df = pd.read_csv(csv_path)

    df.columns = df.columns.str.strip()

    winners = (

        df.sort_values(["√x (sx) error", "T1 (us)", "T2 (us)"],

                       ascending=[True, False, False])

        ["Qubit"].head(n).tolist()

    )

    log.info("Best physical qubits: %s", winners)

    return winners
```

```
N_Q = 5

N_Q_TOTAL = N_Q * 2

PHYSICAL = best_qubits(CAL_CSV, N_Q_TOTAL)


# Phase Only Oracle. Implements: |a⟩|b⟩ → exp(2πi(a + k b)/32) |a⟩|b⟩

def phase_oracle(qc, a_reg, b_reg):


  # Phase contribution from 'a'

  for i in range(N_Q):

    weight = (1 << i)

    angle = 2 * np.pi * weight / ORDER

    qc.rz(angle, a_reg[i])


  # Phase contribution from 'k * b'

  for i in range(N_Q):

    weight = (K_SECRET * (1 << i)) % ORDER

    angle = 2 * np.pi * weight / ORDER

    qc.rz(angle, b_reg[i])


# Build the Shor-style circuit

def shor_ecdlp_phase_circuit() -> QuantumCircuit:


  a = QuantumRegister(N_Q, "a")

  b = QuantumRegister(N_Q, "b")

  c = ClassicalRegister(N_Q * 2, "c")
```

```
    qc = QuantumCircuit(a, b, c, name="ECDLP_32pts_PhaseOnly")

    # Uniform superposition
    qc.h(a)
    qc.h(b)

    # Phase encoding of a + k b
    phase_oracle(qc, a, b)
    qc.barrier()

    # QFT on both registers
    qc.append(QFT(N_Q, do_swaps=False), a)
    qc.append(QFT(N_Q, do_swaps=False), b)

    qc.measure(a, c[:N_Q])
    qc.measure(b, c[N_Q:])

    return qc

# IBM Runtime execution
service = QiskitRuntimeService(channel="ibm_cloud",
                               token=TOKEN,
                               instance=INSTANCE)

backend = service.backend(BACKEND)
```

```
log.info("Backend → %s", backend.name)

qc_raw = shor_ecdlp_phase_circuit()

trans = transpile(qc_raw,
                  backend=backend,
                  initial_layout=PHYSICAL,
                  optimization_level=3)

log.info("Circuit depth %d, gate counts %s", trans.depth(), trans.count_ops())

sampler = SamplerV2(mode=backend)
job = sampler.run([trans], shots=SHOTS)
result = job.result()

# Classical post-processing
creg_name = trans.cregs[0].name
counts_raw = result[0].data.__getattribute__(creg_name).get_counts()

def bits_to_int(bs):
    return int(bs[::-1], 2)

counts = {(bits_to_int(k[N_Q:]), bits_to_int(k[:N_Q])): v
         for k, v in counts_raw.items()}

top = sorted(counts.items(), key=lambda kv: kv[1], reverse=True)
```

```
# Phase-only success criteria: for ideal phase-only: a should lock to 31 and b should lock to (-k) mod 32.
A_TARGET = ORDER - 1         # 31
B_TARGET = (-K_SECRET) % ORDER # 25

a_counts = {a_val: 0 for a_val in range(ORDER)}
for (a_val, b_val), freq in counts.items():
    a_counts[a_val] += freq

p_a_target = a_counts[A_TARGET] / SHOTS

cond_total = 0
cond_b_target = 0
k_spectrum = {k_val: 0 for k_val in range(ORDER)}

for (a_val, b_val), freq in counts.items():
    if a_val != A_TARGET:
        continue
    cond_total += freq
    if b_val == B_TARGET:
        cond_b_target += freq
    k_hat = (-b_val) % ORDER
    k_spectrum[k_hat] += freq

p_b_target_given_a = (cond_b_target / cond_total) if cond_total > 0 else 0.0
```

```
k_sorted = sorted(k_spectrum.items(), key=lambda kv: kv[1], reverse=True)
k_best, k_best_count = k_sorted[0]

found_k = (k_best == K_SECRET)

print(f"\nShots: {SHOTS}")
top_ab, top_ct = top[0]
print(f"Top (a,b): {top_ab} count: {top_ct}")
print(f"P(a={A_TARGET}) = {p_a_target:.4f}")
print(f"P(b={B_TARGET} | a={A_TARGET}) = {p_b_target_given_a:.4f}")
print(f"Best k̂ by peak: {k_best} count: {k_best_count}")

if found_k:
   print(f"\nSUCCESS — k = {K_SECRET} is the dominant phase-only key peak\n")
else:
   print(f"\nWARNING — k = {K_SECRET} is NOT the dominant phase-only key peak\n")

print("Top 10 k̂ values from phase-only key spectrum (conditioned on a=31):")
for k_val, ct in k_sorted[:10]:
   tag = " <<<" if k_val == K_SECRET else ""
   print(f"  k̂ = {k_val:2}   (count = {ct}){tag}")

# Save raw data
out = {
   "experiment": "ECDLP_32pts_PhaseOnly",
    "backend": backend.name,
    "physical_qubits": PHYSICAL,
    "shots": SHOTS,
    "counts": counts_raw
}

JSON_PATH = "FILE_PATH_TO_SAVE_BACKEND_RESULT_JSON.json"

with open(JSON_PATH, "w") as fp:
    json.dump(out, fp, indent=4)

log.info("Results saved → %s", JSON_PATH)

# End

# Code for all visuals from experiment JSON
# Imports
import json
import numpy as np
import matplotlib.pyplot as plt

FILE_PATH = "FILE_PATH_TO_IMPORT_BACKEND_RESULT_JSON.json"

ORDER = 32
N_Q = 5
```

```
A_DOM    = 31
A_TARGET = 31
B_TARGET = 25
K_TRUE   = 7

def bits_to_int(bs: str) -> int:
    return int(bs[::-1], 2)

# Load and build C[a,b]
with open(FILE_PATH, "r") as f:
    data = json.load(f)

counts_raw = data["counts"]

C = np.zeros((ORDER, ORDER), dtype=np.int64)
for bitstring, v in counts_raw.items():
    a_val = bits_to_int(bitstring[N_Q:2*N_Q])
    b_val = bits_to_int(bitstring[:N_Q])
    C[a_val, b_val] += int(v)

shots = int(C.sum())
print(f"Shots: {shots}")

Pab = C / shots
Pa = Pab.sum(axis=1, keepdims=True)
Pb = Pab.sum(axis=0, keepdims=True)
```

```
# Heatmap
plt.figure()
plt.imshow(np.log10(C + 1), origin="lower")
plt.title("Phase-Only Output Heatmap: log10(C[a,b]+1)")
plt.xlabel("b")
plt.ylabel("a")
plt.colorbar(label="log10(count+1)")
plt.show()

# Marginal P(a)
marg_a = C.sum(axis=1)
plt.figure()
plt.bar(range(ORDER), marg_a)
plt.title("Marginal P(a) from Phase-Only Run")
plt.xlabel("a")
plt.ylabel("counts")
plt.show()

# Conditional slice at a=31
row = C[A_DOM, :]
row_total = int(row.sum())

if row_total > 0:
   plt.figure()
   plt.bar(range(ORDER), row)
```

```
    plt.title(f"Conditional Slice: P(b | a={A_DOM}) (counts within a={A_DOM})")
    plt.xlabel("b")
    plt.ylabel(f"counts (within a={A_DOM})")
    plt.show()

# Key spectrum (phase-only)
k_spec = np.zeros(ORDER, dtype=np.int64)
if row_total > 0:
    for b in range(ORDER):
        k_hat = (-b) % ORDER
        k_spec[k_hat] += int(row[b])

plt.figure()
plt.bar(range(ORDER), k_spec)
plt.title(f"Key Spectrum (Phase-Only): k̂ = (-b) mod 32, conditioned on a={A_DOM}")
plt.xlabel("k̂")
plt.ylabel(f"counts (within a={A_DOM})")
plt.show()

# Data Prints
top_idx = np.unravel_index(np.argmax(C), C.shape)
a_top, b_top = top_idx
c_top = C[a_top, b_top]

print(f"Top (a,b): ({a_top}, {b_top}) count: {c_top}")
```

```
p_a_dom = marg_a[A_DOM] / shots
p_b_top_given_a = C[A_DOM, b_top] / row_total

print(f"P(a={A_DOM}) = {p_a_dom:.4f}")
print(f"P(b={b_top} | a={A_DOM}) = {p_b_top_given_a:.4f}")

best_k = int(np.argmax(k_spec))
print(f"Best k̂ by peak (phase-only rule) = {best_k}  count: {int(k_spec[best_k])}")

# Lorenz and Gini
p_flat = Pab.flatten()
p_sorted = np.sort(p_flat)
cum_p = np.cumsum(p_sorted)
n = len(p_sorted)
x = np.arange(1, n + 1) / n

lorenz_x = np.concatenate(([0.0], x))
lorenz_y = np.concatenate(([0.0], cum_p))

area = np.trapezoid(lorenz_y, lorenz_x)
gini = 1.0 - 2.0 * area

plt.figure()
plt.plot(lorenz_x, lorenz_y)
plt.plot([0, 1], [0, 1])
plt.title(f"Lorenz Curve of P(a,b) (Gini ≈ {gini:.4f})")
plt.xlabel("Fraction of outcomes (sorted)")
plt.ylabel("Cumulative probability mass")
plt.show()

print(f"Gini concentration index ≈ {gini:.6f}  (0=uniform, 1=all mass in one cell)")

# PMI
eps = 1e-15
den = np.maximum(Pa @ Pb, eps)
ratio = np.where(Pab > 0, Pab / den, np.nan)
PMI = np.log2(ratio)

plt.figure()
plt.imshow(PMI, origin="lower", aspect="auto")
plt.title("PMI Heatmap: log2(P(a,b)/(P(a)P(b)))")
plt.xlabel("b")
plt.ylabel("a")
plt.colorbar(label="PMI (bits)")
plt.show()

mask = Pab > 0
MI = float(np.sum(Pab[mask] * np.log2(np.maximum(Pab[mask] / den[mask], eps))))
print(f"Mutual Information MI(A;B) ≈ {MI:.6f} bits")

# Key Projection Map
Kmat = np.zeros((ORDER, ORDER), dtype=np.int64)
```

```
for a in range(ORDER):
    for b in range(ORDER):
        k_hat = (-b) % ORDER
        Kmat[a, k_hat] += C[a, b]

plt.figure()
plt.imshow(np.log10(Kmat + 1), origin="lower", aspect="auto")
plt.title("Key-Projection Map: log10(Kmat[a,k̂]+1), where k̂=(-b) mod 32")
plt.xlabel("k̂")
plt.ylabel("a")
plt.colorbar(label="log10(count+1)")
plt.show()

kcol = Kmat[:, K_TRUE]
print(f"Total mass at k̂={K_TRUE}: {int(kcol.sum())}  ({kcol.sum()/shots:.4f} of shots)")
print(f"Mass at (a=31, k̂=7): {int(Kmat[A_TARGET, K_TRUE])}")

# Toroidal Blur (ring mass)
def torus_dist(a, b, a0, b0, mod=32):
    da = min((a - a0) % mod, (a0 - a) % mod)
    db = min((b - b0) % mod, (b0 - b) % mod)
    return float(np.sqrt(da*da + db*db))

D = np.zeros((ORDER, ORDER))
for a in range(ORDER):
    for b in range(ORDER):
        D[a, b] = torus_dist(a, b, A_TARGET, B_TARGET, ORDER)

dvals = D.flatten()
pvals = Pab.flatten()

dmax = int(np.ceil(dvals.max()))
mass_by_r = np.zeros(dmax + 1)

for d, p in zip(dvals, pvals):
    r = int(np.floor(d + 1e-12))
    mass_by_r[r] += p

plt.figure()
plt.bar(np.arange(len(mass_by_r)), mass_by_r)
plt.title("Toroidal Blur Spectrum Around Ideal Peak (31,25)")
plt.xlabel("Radius bucket r = floor(distance)")
plt.ylabel("Probability mass in ring")
plt.show()

print(f"P(ideal peak a=31,b=25) = {C[A_TARGET,B_TARGET]/shots:.6f}
(count={int(C[A_TARGET,B_TARGET])})")
print(f"Mass within r<=1: {mass_by_r[:2].sum():.6f}")
print(f"Mass within r<=2: {mass_by_r[:3].sum():.6f}")

# Per-row Key Fidelity
fidelity = np.zeros(ORDER)
```

```
row_mass = C.sum(axis=1)

for a in range(ORDER):
    if row_mass[a] > 0:
        fidelity[a] = C[a, (-K_TRUE) % ORDER] / row_mass[a]

plt.figure()
plt.bar(range(ORDER), fidelity)
plt.title("Per-row Key Fidelity: P(k̂=7 | a) = C[a,25] / sum_b C[a,b]")
plt.xlabel("a")
plt.ylabel("P(k̂=7 | a)")
plt.show()

print(f"P(k̂=7 | a=31) = {fidelity[A_TARGET]:.6f}")

# SNR curve
dmax = int(np.ceil(max(
    torus_dist(a, b, A_TARGET, B_TARGET, ORDER)
    for a in range(ORDER) for b in range(ORDER)
)))

signal = np.zeros(dmax + 1, dtype=np.float64)

for r in range(dmax + 1):
    s = 0.0
    for a in range(ORDER):
        for b in range(ORDER):
            if torus_dist(a, b, A_TARGET, B_TARGET, ORDER) <= r + 1e-12:
                s += Pab[a, b]
    signal[r] = s

noise = 1.0 - signal
snr = np.where(noise > 0, signal / noise, np.nan)

plt.figure()
plt.plot(range(dmax + 1), snr, marker="o")
plt.title("Peak-vs-Halo SNR on Torus: SNR(r) = mass(d<=r) / mass(d>r)")
plt.xlabel("radius r")
plt.ylabel("SNR(r)")
plt.show()

print(f"SNR(r=0) = {snr[0]:.6f}  (pure peak vs rest)")
print(f"SNR(r=1) = {snr[1]:.6f}")
print(f"SNR(r=2) = {snr[2]:.6f}")

# Edge Map
def wrap(i, mod=32):
    return i % mod

G = np.zeros_like(C)
for a in range(ORDER):
    for b in range(ORDER):
        ap = wrap(a + 1)

        am = wrap(a - 1)

        bp = wrap(b + 1)

        bm = wrap(b - 1)

        G[a, b] = abs(int(C[ap, b]) - int(C[am, b])) + \

                  abs(int(C[a, bp]) - int(C[a, bm]))


plt.figure()

plt.imshow(np.log10(G + 1), origin="lower")

plt.title("Interference Edge Map: log10(|Δa|+|Δb| + 1) (noise-sharpness proxy)")

plt.xlabel("b")

plt.ylabel("a")

plt.colorbar(label="log10(edge+1)")

plt.show()


# Hamming Diffusion

a_star, b_star = np.unravel_index(np.argmax(C), C.shape)

print(f"Dominant peak (a*,b*) = ({a_star},{b_star}) count={int(C[a_star,b_star])}")


def popcount(x):

    return int(bin(x).count("1"))


hd_mass = np.zeros(2 * N_Q + 1)


for a in range(ORDER):

    for b in range(ORDER):
```

```
    hd = popcount(a ^ a_star) + popcount(b ^ b_star)
    hd_mass[hd] += Pab[a, b]

plt.figure()
plt.bar(range(len(hd_mass)), hd_mass)
plt.title("Hamming-Diffusion Spectrum around Dominant Peak (a*,b*)")
plt.xlabel("Hamming distance hd = wt(a⊕a*) + wt(b⊕b*)")
plt.ylabel("Probability mass")
plt.show()

print(f"Mass at hd=0 (exact peak) = {hd_mass[0]:.6f}")
print(f"Mass at hd<=1 = {hd_mass[:2].sum():.6f}")
print(f"Mass at hd<=2 = {hd_mass[:3].sum():.6f}")

# End
```

References

Aleksandrowicz, G., Alexander, T., Barkoutsos, P. Kl., Bello, L., Ben-Haim, Y., Bucher, D., Cabrera-Hernández, F. J., Carballo-Franquis, J., Chen, A., Chen, C.-F., Chow, J. M., Córcoles-Gonzales, A. D., Cross, A. J., Cross, A. W., Cruz-Benito, J., Culver, C., De La Puente González, S., De La Torre, E., Ding, D., ... Zoufal, C. (2019). *Qiskit: An Open-source Framework for Quantum Computing*. https://doi.org/10.5281/ZENODO.2562111

Galbraith, S. D. (2012). *Mathematics of Public Key Cryptography*. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9781139012843

Shor, P. W. (1997). *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer.* SIAM Journal on Computing, 26(5), 1484-1509. https://doi.org/10.1137/S0097539795293172