

Supplementary material: *Answer Partitions and Oracle Access* *Determine Quantum Query Complexity* Detailed guide to the finite partition census

Karl Svozil

21 July 2026

1 Purpose and scope

This supplement documents the Python implementation used for the finite balanced-partition census in the accompanying manuscript. The program does not simulate a particular hardware circuit. It compares two query models for the same finite oracle family:

1. deterministic, exact, adaptive classical value queries; and
2. exact, zero-error quantum queries to the reversible addition oracle

$$O_a |x, y\rangle = |x, y + f[a, x] \pmod d\rangle.$$

The central methodological rule is that a partition is a task specification, whereas the oracle determines which distinctions one call can make. The program therefore holds the oracle table fixed while varying the balanced two-cell partition of its hidden instances. Numerical SDP results are labelled as candidates; they are never silently promoted to exact theorems.

The archive contains the Python source, tests, requirements, generated CSV files, representative witness files, a Lean 4 companion formalization, and this document. The main scanner source is `code/partition_query_scan.py`; the formal source is under `lean/`.

2 Input model

An oracle family is an integer matrix $f[a, x]$. Row a denotes one possible hidden instance and column x denotes one permitted query address. The response alphabet is $\mathbb{Z}_d$, and a query is the reversible addition oracle

$$O_a |x, y\rangle = |x, y + f[a, x] \pmod d\rangle. \quad (1)$$

For the built-in Boolean family with `-boolean-inputs M`, the rows are all 2^M binary strings and the columns are the M coordinates. A balanced binary task assigns half the rows to answer cell 0 and half to cell 1. Interchanging the two answer names does not change the task, so the number of unlabeled partitions for $n = 2k$ instances is

$$M_n = \frac{1}{2} \binom{n}{n/2}. \quad (2)$$

The implementation represents a cell by a Python integer bit mask. To avoid double counting, `balanced_partitions` always puts instance 0 in cell 0. It can either enumerate all representatives or draw a reproducible uniform sample using `random.Random(seed)`.

Custom tables can be supplied as CSV or JSON. CSV rows may contain only integer responses, or an instance name followed by the responses. JSON may be a matrix, or an object with `values`, `names`, and optionally `alphabet_size`. The loader validates equal row widths, nonnegative responses, unique names, and a response alphabet containing every response.

3 Classical exact complexity

For each partition, `deterministic_exact_queries` computes the optimal worst-case depth of an adaptive classical decision tree. A state of the dynamic program is the set of still possible instances, represented by an integer mask. A state has cost zero when all remaining instances lie in one answer cell. Otherwise the routine tries every query address, forms one child mask for each observed response, and applies the recurrence

$$D(S) = 1 + \min_x \max_{v: S_{x,v} \neq \emptyset} D(S_{x,v}). \quad (3)$$

Memoization makes each candidate mask appear only once. If no query separates the cells, the return value is `None`; this is an operationally impossible task for the supplied value interface, not a large finite cost. The resulting `D_exact` column is an exact classical quantity.

4 Quantum one-query and multi-query tests

4.1 Fourier modes of the response

The response register is diagonalized in its Fourier basis. The idle mode is included, and each nontrivial mode $q = (x, s)$ has phase vector

$$v_q(a) = \exp(2\pi i s f[a, x]/d), \quad \Delta_q(a, b) = \overline{v_q(a)} v_q(b). \quad (4)$$

The function `query_mode_matrices` builds these matrices. For a Boolean response, there is one nontrivial phase mode per address and $v_x(a) = (-1)^{f[a, x]}$. For a non-Boolean cyclic response, all nonzero Fourier frequencies are retained.

4.2 Exact Boolean one-query test

For $d = 2$ and one query, the Gram constraints reduce to a finite rational linear feasibility problem. A weight w_q is assigned to the idle mode and to each address phase mode. The weights obey $w_q \geq 0$, $\sum_q w_q = 1$, and for every cross-cell pair a, b ,

$$\sum_q w_q \Delta_q(a, b) = 0. \quad (5)$$

`exact_one_query_boolean_weights` deduplicates identical constraint rows, computes ranks and candidate bases with exact `fractions.Fraction` arithmetic, and accepts only a nonnegative exact solution. Thus one-query feasibility and infeasibility for Boolean tables are not floating-point classifications.

4.3 Gram-matrix SDP for a fixed query count

For general T , `exact_quantum_feasible` implements the Gram-matrix evolution SDP. It introduces positive semidefinite matrices $M[t, q]$, one for each query level and Fourier mode, with constraints

$$\sum_q M[0, q] = J, \quad (6)$$

$$\sum_q M[t+1, q] = \sum_q \Delta_q \odot M[t, q], \quad (7)$$

$$G_{\text{final}} = \sum_q \Delta_q \odot M[T-1, q], \quad (8)$$

where J is the all-ones Gram matrix and $\odot$ is entrywise multiplication. Exact discrimination requires

$$G_{\text{final}}[a, b] = 0 \quad \text{whenever the labels of } a \text{ and } b \text{ differ.} \quad (9)$$

CVXPY supplies the numerical SDP backend. The program records the selected solver and its raw status. For a returned feasible point, `_audit_sdp_solution` checks positive semidefiniteness, every Gram recurrence, the initial condition, and the cross-cell zeros. The largest violation is stored as `max_residual`. A floating-point feasible point is still labelled `numerical_candidate`, because approximate feasibility does not by itself prove the existence of an exact feasible Gram system; numerical infeasibility likewise does not prove a lower bound.

5 Exact constructive upper bounds

The ordinary classical decision tree is automatically a quantum upper bound: the quantum protocol can query the same basis values. For binary table entries and an even response alphabet, the response shift has an order-two Fourier character. One Deutsch query can then obtain

$$f[a, x_i] \oplus f[a, x_j] \quad (10)$$

for any pair of addresses. The routine `pair_parity_decision_tree` searches an adaptive tree whose tests are single values or pair parities. Every tree node costs one oracle call, so a tree of depth T is an explicit exact upper certificate $Q \leq T$.

The routine `exact_quantum_queries` first obtains this constructive upper bound, then tests only smaller query counts. If all smaller counts are excluded exactly, the result is certified `exact`. If a smaller count is excluded by a numerical SDP, the result is retained as `numerical_candidate`. A user-supplied `-max-quantum-queries` cap can stop the lower-bound search early; such rows are marked `cap_unresolved`.

6 Symmetry reduction

For the Boolean cube, `boolean_cube_automorphisms` generates the oracle-preserving transformations obtained by coordinate permutations and independent coordinate complements. The answer-cell exchange is then quotiented as well. These transformations preserve query cost: they only rename addresses or apply known response relabelings. General linear changes of coordinates are deliberately not used, since they can turn a single value query into a parity query and therefore need not preserve the access model.

`balanced_boolean_partition_orbits` computes one canonical cell mask per orbit and records its orbit size. Orbit-weighted summaries recover counts of the original partitions:

Boolean addresses	instances	balanced partitions	orbits
3	8	35	6
4	16	6,435	58

Orbit reduction reduces SDP calls but does not remove the full-population enumeration needed to form the orbits. It cannot be combined with uniform partition sampling.

7 Witness files and CSV output

Each CSV row contains the partition mask and cells, orbit size, classical depth, quantum lower and upper bounds, candidate speedup, solver information, certification level, residual, and the associated witness path. The columns have the following interpretation:

D_exact exact deterministic adaptive classical depth;

Q_exact_candidate exact quantum depth when certified, or the best candidate when numerical lower bounds are involved;

Q_lower_bound the largest lower-bound count established by the completed search;

Q_constructive_upper_bound depth of the stored classical or value/parity tree;

certification exact, numerical_candidate, unresolved, or an impossibility/cap status;

max_residual audited numerical violation, when applicable.

With `-witness-dir`, exact Boolean one-query results store rational Fourier weights, exact upper bounds store the complete decision tree, and feasible numerical SDPs store compressed Gram matrices in `.npz` files plus JSON metadata. In orbit mode the witness is for the canonical representative; `orbit_size` gives its multiplicity.

8 Reported census and certification status

The three-address scan is exact for all 35 balanced partitions. Its weighted histogram is

(D, Q)	(1,1)	(2,1)	(2,2)	(3,2)
partitions	3	3	12	17

Thus 20 of 35 tasks have an exact advantage: three by a factor of two and 17 by a factor of $3/2$. The witness for the representative cell $\{000, 001, 010, 100\}$ is the two-query majority construction: obtain $b_1 \oplus b_2$, then query b_1 if that result is zero and b_3 if it is one.

For four addresses, exact one-query lower obstructions and exact constructive trees settle 395 of 6,435 partitions:

(D, Q)	(1,1)	(2,1)	(2,2)	(3,2)	(4,2)
exactly certified	4	6	48	248	89

For the other 43 orbit representatives, covering 6,040 partitions, the two-query SDP was numerically reported infeasible by SCS at requested tolerances 10^{-7} and 10^{-8} , while an exact value/parity tree supplied a three-query upper bound. If those numerical lower bounds are correct, the corresponding partitions comprise 1,536 candidates with $(D, Q) = (3, 3)$ and 4,504 with $(D, Q) = (4, 3)$. They remain numerical candidates in the manuscript and in this archive. A tighter tolerance is a stability check, not an exact dual certificate.

9 Formal verification in Lean 4

The archive includes a companion project in `lean/`, checked with Lean 4.30.0 and `mathlib` v4.30.0 [3, 4]. Its purpose is not to reimplement the floating-point SDP solver. It machine-checks the exact algebraic layer and executes independent finite decision procedures for the exact census claims. The umbrella module is `lean/AnswerPartitions.lean`.

The ordinary kernel-checked proofs cover answer fibers, the equivalence between cross-cell pure-state Gram zeros and orthogonal output-cell spans, fixed unitary invariance, the Deutsch Gram pattern, controlled-response sufficiency, and the cyclic-response condition that a positive-dimensional shift has a -1 eigenvalue exactly when its dimension is even. They also prove the Boolean weighted-distance condition and its essential-coordinate obstruction, four-bit truth-table/mask equivalences, and the finite linear-algebraic core of the Bernstein–Vazirani discussion.

input size	deterministic-depth histogram	certificate-depth histogram
3	1:3, 2:15, 3:17	1:6, 2:29
4	1:4, 2:54, 3:1784, 4:4593	1:10, 2:385, 3:6040

Table 1: Lean census results over all 35 normalized balanced three-bit tasks and all 6,435 normalized balanced four-bit tasks.

The executable census results are summarized in Table 1. Here “certificate depth” means the formally defined adaptive tree whose nodes query either one bit or one pair parity.

For three inputs, Lean also checks that exactly 20 tasks have certificate depth below deterministic depth and verifies the six declared symmetry orbits. For four inputs, it checks the ten depth-one certificate rows $(D, C) = (1, 1) : 4, (2, 1) : 6$ and the 385 further depth-two certificate rows $(2, 2) : 48, (3, 2) : 248, (4, 2) : 89$, together with a formal one-query weight obstruction for those 385 cases. Here C denotes the encoded certificate depth. These computations agree with the exact portion of the Python census.

There are two qualifications. First, the three closed census audits use `native_decide`; these theorems therefore trust Lean’s compiler and native evaluator in addition to its proof kernel. The remaining displayed algebraic proofs are ordinary proof terms. The delivered source contains no handwritten `sorry`, `admit`, or project-declared axiom. Second, the Lean project does not yet define unrestricted quantum-query semantics or prove the bridge that realizes each pair-parity node by one Deutsch query and composes such nodes coherently. Thus the finite tree counts are theorem-grade statements about the encoded certificate model; their interpretation as unrestricted Q_E additionally uses the analytic argument in the manuscript.

In particular, the restricted certificate-depth count of 6,040 does not formalize the numerical two-query SDP lower bounds. The 43 corresponding orbits remain numerical candidates, exactly as elsewhere in this supplement. No numerical infeasibility claim has been promoted to a Lean theorem.

To reproduce the formal checks from the extracted archive root, use:

```
cd lean
lake update
lake build
```

The toolchain file pins Lean 4.30.0 and the Lake configuration pins mathlib v4.30.0. Build products, dependency caches, and machine-local paths are not included. The claim-by-claim coverage and trust report is `lean/FINDINGS.md`.

10 Cross-validation against published NPN data

Appendix A of Montanaro, Jozsa, and Mitchison [2] reports numerical success probabilities for Boolean functions under NPN equivalence. We mapped each of our 58 orbit representatives to their integer truth-table identifier. Six representatives contain dummy variables and reduce to their one-, two-, or three-bit tables. Among the 52 nondegenerate balanced four-bit representatives, the nine that our exact certificates place at two queries have published two-query entry (1). For all 43 remaining representatives, their reported two-query success probability is below one, ranging from 0.900 to 0.986. Thus the two computations agree on every balanced orbit.

The complete crosswalk is in `data/m4-mjm-crosscheck.csv`. The columns `mjm_one_query`, `mjm_two_query`, and `mjm_three_query` transcribe the published table; an asterisk denotes their nonadaptive construction. This is an independent check of the orbit enumeration and numerical classification, not an exact infeasibility certificate. Both analyses explicitly caution that a floating-point entry below or near one does not prove exact query complexity.

11 Reproduction commands

Run these commands from the root of the extracted archive. Python 3.10 or later is recommended.

```
python -m pip install -r code/requirements-partition-scan.txt

python code/partition_query_scan.py --boolean-inputs 3 --orbit-reduce \\
--show-all --csv data/results-m3-orbits.csv \\
--witness-dir witnesses-m3

python code/partition_query_scan.py --boolean-inputs 4 --orbit-reduce \\
--solver SCS --tolerance 1e-8 \\
--csv data/results-m4-orbits.csv \\
--witness-dir witnesses-m4
```

The four-address run is the expensive one. To inspect only the exact one-query screen, use:

```
python code/partition_query_scan.py --boolean-inputs 4 --orbit-reduce \\
--max-quantum-queries 1 --csv data/results-m4-one-query.csv
```

For a small smoke test:

```
python code/partition_query_scan.py --boolean-inputs 2 --show-all
python -m unittest discover -s code -p 'test*.py' -v
```

The exact output can depend on installed solver versions and numerical tolerances. The archive preserves the original CSV files and witness metadata used for the manuscript; the four-address numerical status should be rechecked independently before being quoted as a theorem.

12 Limitations

The scan concerns balanced binary partitions, exact zero-error query complexity, and one specified reversible value-oracle interface. It does not measure gate count, oracle-construction cost, bounded-error complexity, noise, or asymptotic families. The finite proportions are therefore descriptive finite-model data. The next rigorous step for the four-address candidates is to extract exact primal or dual certificates for the two-query SDPs.

References

- [1] H. Barnum, M. Saks, and M. Szegedy, “Quantum query complexity and semi-definite programming,” in *Proc. 18th IEEE Conference on Computational Complexity*, 179–193 (2003), doi:10.1109/CCC.2003.1214419.
- [2] A. Montanaro, R. Jozsa, and G. Mitchison, “On exact quantum query complexity,” *Algorithmica* **71**, 775–796 (2015), doi:10.1007/s00453-013-9826-8.
- [3] L. de Moura and S. Ulrich, “The Lean 4 theorem prover and programming language,” in *Automated Deduction—CADE 28*, Lecture Notes in Computer Science **12699**, 625–635 (Springer, 2021), doi:10.1007/978-3-030-79876-5_37.
- [4] The mathlib Community, “The Lean mathematical library,” in *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, 367–381 (ACM, 2020), doi:10.1145/3372885.3373824.