Algebraic Entropy and Conditional Mutual Information in a Tiny Gauge-Invariant Truncated Hilbert Space: A Reproducible Toy-Model Study with Effective Mixing Hamiltonians

Lluis Eriksson
Independent Researcher
lluiseriksson@gmail.com

January 2026

Abstract

We present a fully reproducible Google Colab pipeline to compute region algebraic entropies and conditional mutual informations (CMI) in a tiny truncated Hilbert space (here dim = 8) indexed by discrete fusion-like descriptors desc = (x, μ) on $L = 4$ cells. To generate nontrivial ground states within the descriptor-labeled subspace, we introduce an effective Hermitian mixing Hamiltonian based on a weighted k-nearest-neighbor (kNN) graph Laplacian over configuration labels. Across a parameter sweep, we identify a strong-mixing regime where the participation ratio approaches dim (consistent with Laplacian-dominated ground states on connected graphs) and algebraic CMI diagnostics become extremely small (down to 10^{-6} and below) for the chosen algebraic factorization, while region algebraic entropies remain $O(1)$ and exhibit near-quantized values $\approx n \log 2$. We stress that the mixing term is an ansatz used to probe information-theoretic diagnostics and is not claimed to coincide with a Kogut–Susskind plaquette operator. All artifacts (CSV/JSON, figures, and model dumps) are generated in one run and packaged as Overleaf-ready assets.

1 Scope, claims, and contributions

This note is intentionally narrow in scope. It does *not* attempt a physical construction of lattice Yang–Mills dynamics, nor any continuum/thermodynamic scaling analysis. Instead, it addresses a simpler, operational question:

Given a fixed small gauge-invariant truncated Hilbert space and a specified algebraic factorization derived from discrete labels (x, μ), can we generate nontrivial ground states and quantitatively diagnose algebraic entanglement and conditional independence via algebraic entropy and algebraic CMI?

Concretely, we contribute:

1. A one-shot reproducible pipeline to compute algebraic entropies $S_{\mathrm{alg}}(R)$ and algebraic CMI diagnostics $I(A : C \mid B)$ from a gauge-invariant basis indexed by (x, μ).
2. A simple, tunable effective Hamiltonian family $H = H_E + t_{\mathrm{mix}} L$ acting entirely within the gauge-invariant subspace (Section 5).
3. A numerical sweep showing that in a strong-mixing regime the ground state becomes close to a uniform superposition (PR $\approx$ dim), while the resulting algebraic CMI diagnostics are near-zero within numerical precision for the chosen factorization.

Careful interpretation. When we refer to "approximately Markovian" behavior, we mean an *operational* statement: the computed algebraic CMI values are very small with respect to the specific algebraic partition induced by (x, μ) and the implemented definition of S_{alg}. We do not claim a general quantum Markov property in the usual tensor-product sense, nor any universality beyond this toy setting.

2 Truncated gauge-invariant Hilbert space and labels

We work in a fixed truncated Hilbert space with orthonormal basis $\{|i\rangle\}_{i=1}^{\dim}$, where each basis vector is indexed by a descriptor $\text{desc} = (x, \mu)$:

- $x = (x_1, x_2, x_3)$ encodes boundary/cut labels between adjacent cells. Concretely, each $x_s = (x_s^{\text{top}}, x_s^{\text{bot}}) \in \{0, 1\}^2$ stores two binary cut labels.
- $\mu = (\mu_1, \mu_2, \mu_3, \mu_4)$ encodes local discrete degrees of freedom per cell; each μ_p is a (possibly multi-bit) tuple.

We take $L = 4$ cells and, for the dataset studied here, $\dim = 8$.

Terminology. We refer to the basis as "gauge-invariant" in the operational sense that it is intended to represent constraint-compatible (Gauss-law-like) configurations in a reduced description; however, in this note we do not derive (x, μ) from an explicit $SU(2)$ Kogut–Susskind truncation.

Logarithms. All entropies use natural logarithms, so the unit is *nats*. Thus $\log 2 \approx 0.6931$.

3 Regions and target information-theoretic quantities

We consider contiguous regions $R \subset \{1, 2, 3, 4\}$ and compute algebraic entropies $S_{\text{alg}}(R)$. We define two conditional mutual informations:

$$I_{w=1} = I(A : C \mid B), \quad A = \{1\},\ B = \{2\},\ C = \{3\},$$

$$I_{w=2} = I(A : C \mid B), \quad A = \{1\},\ B = \{2, 3\},\ C = \{4\}.$$

We also report

$$I_{\text{sum}} = I_{w=1} + I_{w=2}.$$

Purity sanity check. For a pure state $|\psi\rangle$ on the full system, we enforce $S_{\text{alg}}(1..4) \approx 0$ as a numerical check.

4 Definition of algebraic entropy and algebraic CMI (as implemented)

Let $|\psi\rangle = \sum_{i=1}^{\dim} \psi_i |i\rangle$ be a normalized pure state in the gauge-invariant basis. For a region $R = [i..j]$, we define a discrete boundary label $\alpha = \alpha(\text{desc}; i, j)$ (a tuple extracted from x at cut boundaries) which indexes superselection sectors for the algebraic factorization.

Within each sector α, basis states are grouped into a region key k_R and complement key k_C (deterministic functions of (x, μ), defined identically to the accompanying code). We build a complex matrix M_α whose entries sum amplitudes of basis states mapping to the same (k_R, k_C) pair:

$$(M_\alpha)_{r,c} = \sum_{\text{desc} \mapsto (\alpha, r, c)} \psi(\text{desc}).$$

Let $\{s_{\alpha,\ell}\}_\ell$ be the singular values of M_α, and define weights $w_{\alpha,\ell} = s_{\alpha,\ell}^2$. The algebraic entropy is

$$S_{\mathrm{alg}}(R) \;=\; -\sum_{\alpha} \sum_{\ell:\; w_{\alpha,\ell}>\varepsilon} w_{\alpha,\ell} \log w_{\alpha,\ell},$$

where a small cutoff ε discards numerical zero modes only.

Normalization check. For normalized $|\psi\rangle$, we numerically verify

$$\sum_{\alpha,\ell} w_{\alpha,\ell} = 1$$

within a fixed tolerance.

Algebraic CMI diagnostics. We compute

$$I(A:C \mid B) = S_{\mathrm{alg}}(AB) + S_{\mathrm{alg}}(BC) - S_{\mathrm{alg}}(B) - S_{\mathrm{alg}}(ABC).$$

In the usual tensor-product setting, CMI is nonnegative by strong subadditivity. In our setting S_{alg} is defined by an explicit sector/block coarse-graining (Section 4), so nonnegativity is an empirical diagnostic rather than a guaranteed theorem. In our sweeps we observe $I(A:C \mid B) \geq -10^{-12}$; values below this threshold would indicate either numerical issues or a genuine violation of strong subadditivity for the chosen coarse-graining. For log-scale plots one may use the clipped version

$$I_{\mathrm{clip}} = \max(I, 10^{-16}).$$

5 Effective Hamiltonian family

We study an effective Hamiltonian family acting within the gauge-invariant subspace:

$$H = H_E + t_{\mathrm{mix}}\, L.$$

5.1 Diagonal "electric proxy" term H_E

We define a diagonal proxy H_E as a sum of local contributions derived from the binary label bits. Each bit $b \in \{0,1\}$ is mapped to a cost inspired by $j(j+1)$ for $j \in \{0, \frac{1}{2}\}$:

$$\mathrm{cost}(b) = \begin{cases} 0 & b = 0, \\ \frac{3}{4} & b = 1. \end{cases}$$

Then H_E is defined as $H_E = \mathrm{diag}(E(\mathrm{desc}))$, where $E(\mathrm{desc})$ sums costs over all bits in (x,μ), with an overall prefactor $\frac{g^2}{2}$.

5.2 Mixing term via a weighted kNN graph Laplacian

Let $b(\mathrm{desc}) \in \{0,1\}^m$ be the flattened bitstring for (x,μ). Define pairwise distances by Hamming distance $d(i,j)$ between bitstrings.

We build a weighted kNN graph on the dim configurations: for each node i we connect to its k nearest neighbors, with symmetric edge weights

$$A_{ij} = \exp(-\alpha_w\, d(i,j)),$$

and define the graph Laplacian

$$L = D - A, \qquad D_{ii} = \sum_j A_{ij}.$$

This yields a real symmetric (hence Hermitian) mixing term. The design intent is to generate controllable superpositions in the gauge-invariant basis. No claim is made that L matches a Kogut–Susskind magnetic plaquette term [1].

6 Numerical procedure and reported metrics

For each parameter pair $(\alpha_w, t_{\text{mix}})$ on a fixed grid:

1. Construct H, diagonalize it, and take the ground state $|\psi_0\rangle$.
2. Compute region entropies $S_{\text{alg}}(R)$, CMI diagnostics $I_{w=1}, I_{w=2}$, and I_{sum}.
3. Compute the participation ratio

$$\text{PR}(\psi_0) = \frac{1}{\sum_i |\psi_{0,i}|^4},$$

and the spectral gap proxy

$$\Delta = E_1 - E_0, \qquad \xi_{\text{spec}} = \Delta^{-1} \quad (\Delta > 0).$$

Interpretation of ξ_{spec}. We emphasize that $\xi_{\text{spec}} = \Delta^{-1}$ is a finite-dimensional spectral proxy whose value depends on the overall energy scale, in particular on t_{mix} (large t_{mix} can produce very large Δ and hence very small ξ_{spec}). No claim is made that it corresponds to a physical correlation length in any continuum or infinite-volume limit.

7 Artifacts and Overleaf bundle

The one-shot run generates:

- tabular outputs: `sweep.csv`, `sweep.json`;
- figures: `PR_vs_CMI.png`, `Delta_vs_CMI.png`, `S23_vs_S13.png`;
- selected models: `best_maxPR.json`, `best_minCMI_PRge2.json`;
- eigen-data dumps for selected models: `H_*.npy`, `evals_*.npy`, `psi0_*.npy`;
- an `overleaf_bundle.zip` for direct upload.

8 Results

8.1 Sweep-level plots

Figure 1 shows PR versus I_{sum} across the sweep, and Figure 2 shows Δ versus I_{sum}. Figure 3 shows an entropy-structure scatter.

8.2 Selected models (JSON dumps)

Table 1 reports the two automatically selected models: (i) maximum PR, and (ii) minimum I_{sum} under the constraint PR ≥ 2. For the sweep summarized in the prompt, the strong-mixing point dominates the explored grid and yields PR close to dim with very small CMI.

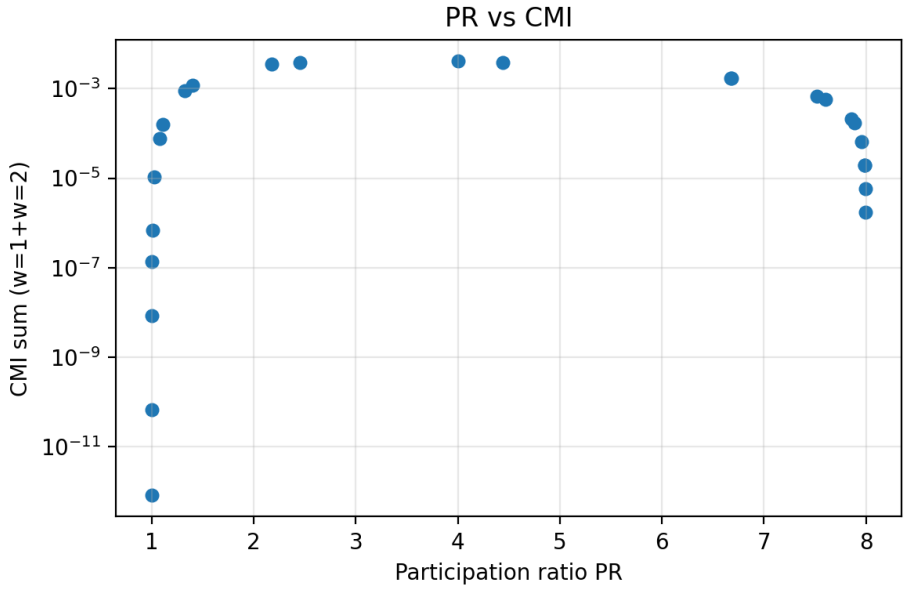

Figure 1: Participation ratio (PR) versus total algebraic CMI $I_{\text{sum}} = I_{w=1} + I_{w=2}$ across the parameter sweep (log scale on CMI).

9 Discussion

9.1 Why PR $\approx$ dim and CMI ≈ 0 occur in the strong-mixing regime

At $\alpha_w = 0$ the kNN adjacency weights satisfy $A_{ij} = 1$ on selected edges, so L becomes the combinatorial graph Laplacian of a regular graph (in the reported run, degree 3). For a connected graph, L has a lowest eigenvalue 0 with eigenvector proportional to the all-ones vector $\mathbf{1}$. Therefore, when t_{mix} dominates the diagonal proxy term H_E, the ground state of $H = H_E + t_{\text{mix}} L$ approaches a near-uniform superposition

$$|\psi_0\rangle \;\approx\; \frac{1}{\sqrt{\dim}} \sum_{i=1}^{\dim} |i\rangle,$$

yielding PR close to dim.

In this regime, the observed near-quantized values $S_{\text{alg}}(R) \approx n \log 2$ and the very small algebraic CMI values are consistent with the implemented sector/block structure induced by the (x, μ)-based algebraic factorization (Section 4). Intuitively, as the ground state approaches a near-uniform superposition in the configuration basis, the induced blockwise Schmidt spectrum tends toward being approximately flat over an effective 2^n set of modes for certain regions, yielding $S_{\text{alg}}(R) \approx n \log 2$. Operationally, this indicates near-conditional-independence *with respect to that algebraic factorization.*

9.2 What this does (and does not) say about "quantum Markov" structure

In standard quantum information theory, exact saturation of strong subadditivity ($I(A : C \mid B) = 0$) implies a quantum Markov structure and is characterized by a recovery map [3]. Our setting differs because:

- entropies are computed via a sector-decomposed algebraic prescription tied to (x, μ) labels,
- the system is extremely small (dim = 8),

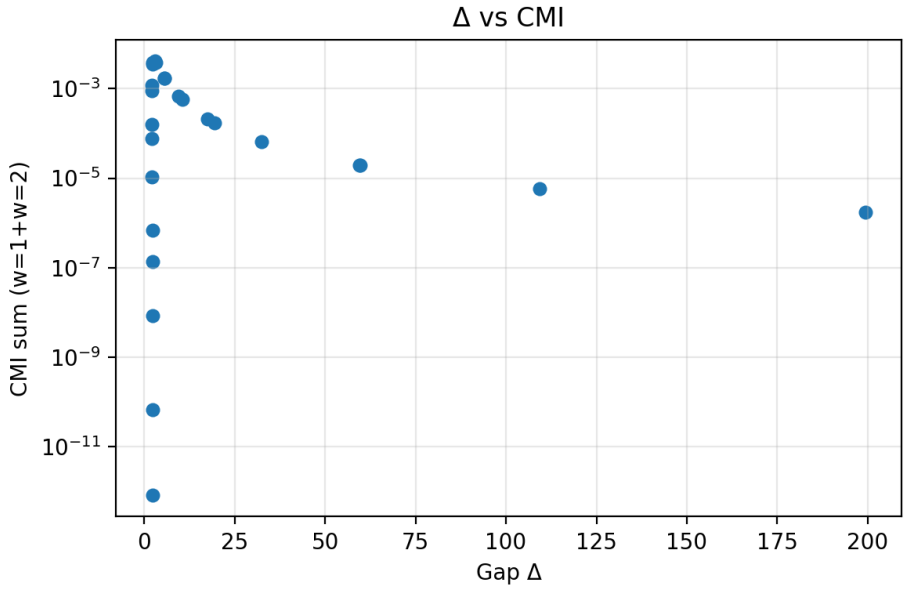

Figure 2: Spectral gap proxy Δ versus total algebraic CMI across the sweep (log scale on CMI).

- the Hamiltonian mixing term is an effective graph Laplacian ansatz.

Thus, small values of the computed algebraic CMI should be read as a diagnostic of conditional-independence in this toy, label-induced algebraic setting, rather than as evidence for a physically universal Markov property.

9.3 Limitations

- **No scaling.** We do not study how observables behave as dim or L increase.
- **Hamiltonian not physical.** The mixing term is not derived from a Kogut–Susskind magnetic term; it is a controllable ansatz.
- **Spectral proxy.** The proxy $\xi_{\text{spec}} = \Delta^{-1}$ is scale-dependent and should not be overinterpreted.

9.4 Recommended additional experiments (straightforward extensions)

These substantially strengthen interpretability without changing the core pipeline:

1. **Baselines.** Compare I_{sum} against (i) Haar-random pure states in $\mathbb{C}^{\text{dim}}$, and/or (ii) ground states of random Hermitian matrices (GOE/GUE). Report empirical distributions of S_{alg} and I_{sum}.
2. **Ablations.** Repeat sweeps for $k_{\text{nn}} \in \{1, 2, 3, \min(5, \dim - 1)\}$ and alternative distance weights (e.g. weighting x-bits versus μ-bits differently) to test robustness.
3. **Finer t_{mix} grid.** Add smaller t_{mix} values (e.g. $0, 0.1, 0.3, 1, 3, 10$) to resolve the transition from diagonal H_E ground states (trivial entropies) to the Laplacian-dominated regime.

10 Numerical precision and stability

Occasional values of CMI at the level of 10^{-16} (including slight negatives) are interpreted as floating-point round-off in double precision (machine epsilon $\varepsilon_{\text{mach}} \approx 2.2 \times 10^{-16}$) accumulated

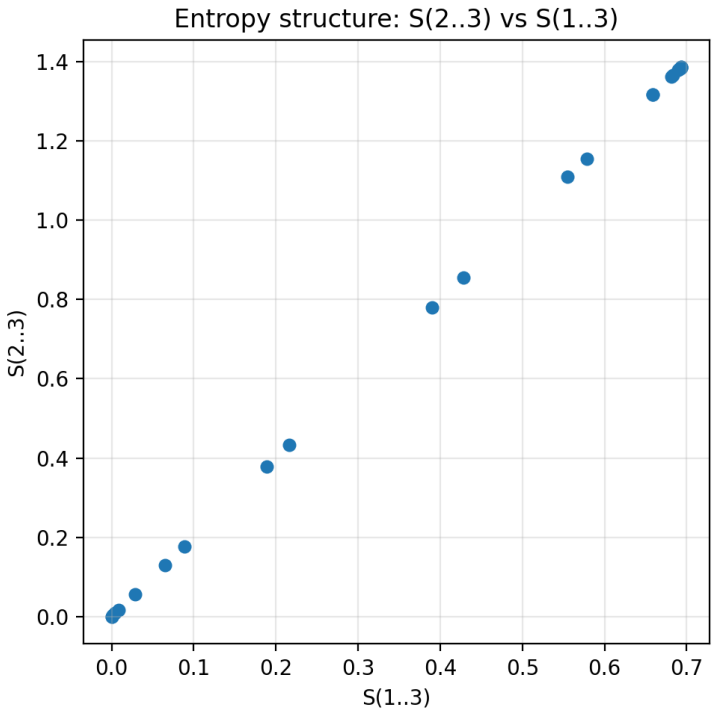

Figure 3: Entropy-structure scatter: $S_{\text{alg}}(2..3)$ versus $S_{\text{alg}}(1..3)$.

over many linear-algebra operations. For log-scale plots, we use $\max(I_{\text{sum}}, 10^{-16})$ to avoid numerical issues, and when reporting aggregated metrics we may optionally use $I_{\text{clip}} = \max(I, 0)$.

11 Reproducibility

A single Google Colab cell (Appendix B) produces a timestamped run folder under `MyDrive/algebraic_entropy_` containing `sweep.csv`, `sweep.json`, all figures, and best-model dumps, and packages the artifacts as `overleaf_bundle.zip`. The script assumes that the basis file `descs.pkl` has been generated previously and is present in `MyDrive/algebraic_entropy_runs/`.

A Minimal pseudocode overview

The workflow is:

1. Load basis descriptors $\{\text{desc}_i\}_{i=1}^{\dim}$ from `descs.pkl`.
2. Build bitstrings $b(\text{desc}_i)$ and the Hamming distance matrix D_{ij}.
3. For each $(\alpha_w, t_{\text{mix}})$:
   (a) build kNN adjacency A and Laplacian L;
   (b) build $H = H_E + t_{\text{mix}} L$;
   (c) diagonalize H and take the normalized ground state $|\psi_0\rangle$;
   (d) compute $S_{\text{alg}}(R)$ for required regions via sectored SVD;
   (e) compute $I_{w=1}$, $I_{w=2}$, PR, Δ, and save to CSV/JSON.

Quantity	Best-by-PR	Best-by-CMI (PR$\geq$ 2)
α_w	0.0	0.0
t_{mix}	100.0	100.0
k_{nn}	3	3
edges	12	12
E_0	2.6179476	2.6179476
Δ	199.4735852	199.4735852
$\xi_{\mathrm{spec}} = \Delta^{-1}$	0.0050132	0.0050132
PR	7.9989229	7.9989229
$S_{\mathrm{alg}}(1..2)$	0.6931399	0.6931399
$S_{\mathrm{alg}}(2..3)$	1.3862379	1.3862379
$S_{\mathrm{alg}}(2)$	1.3862571	1.3862571
$S_{\mathrm{alg}}(1..3)$	0.6931190	0.6931190
$S_{\mathrm{alg}}(2..4)$	0.6931190	0.6931190
$S_{\mathrm{alg}}(1..4)$	2.2×10^{-16}	2.2×10^{-16}
$I_{w=1}$	1.76×10^{-6}	1.76×10^{-6}
$I_{w=2}$	1.39×10^{-11}	1.39×10^{-11}
I_{sum}	1.76×10^{-6}	1.76×10^{-6}

Table 1: Automatically selected models from the sweep (as reported in `paper_summary.txt`).

B Full one-shot Colab script (reproducibility + figures + Overleaf bundle)

Copy-paste the following into a single Google Colab cell and run. It will generate all outputs and download an `overleaf_bundle.zip`.

```
# ============================================================
# ONE-SHOT COLAB RUNNER (reproducibility + artifacts + Overleaf zip)
#
# Generates in a new run folder under MyDrive/algebraic_entropy_runs/<timestamp>/:
#   - sweep.csv, sweep.json
#   - PR_vs_CMI.png, Delta_vs_CMI.png, S23_vs_S13.png
#   - paper_summary.txt
#   - best_maxPR.json, best_minCMI_PRge2.json
#   - H_maxPR.npy, evals_maxPR.npy, psi0_maxPR.npy
#   - overleaf_bundle.zip (downloadable)
#
# Model family: H = H_E(proxy, diagonal) + t_mix * L_kNN(alpha_w)
# ============================================================

import os, json, pickle, csv, zipfile
from datetime import datetime
from collections import defaultdict
import numpy as np
import matplotlib.pyplot as plt

# Drive mount (safe)
if not os.path.isdir("/content/drive/MyDrive"):
    from google.colab import drive
```

```
    drive.mount("/content/drive", force_remount=False)

from google.colab import files

RUNS_DIR = "/content/drive/MyDrive/algebraic_entropy_runs"
os.makedirs(RUNS_DIR, exist_ok=True)

RUN_ID = datetime.now().strftime("%Y%m%d_%H%M%S")
OUTDIR = os.path.join(RUNS_DIR, RUN_ID)
os.makedirs(OUTDIR, exist_ok=True)
print("[info] OUTDIR =", OUTDIR)

def find_latest(runs_dir, filename):
    hits = []
    for name in os.listdir(runs_dir):
        d = os.path.join(runs_dir, name)
        if not os.path.isdir(d):
            continue
        p = os.path.join(d, filename)
        if os.path.exists(p):
            hits.append((name, os.path.getmtime(p), p))
    if not hits:
        return None
    hits.sort(key=lambda t: (t[0], t[1]))
    return hits[-1][2]

DESCS_PKL = find_latest(RUNS_DIR, "descs.pkl")
if DESCS_PKL is None:
    raise RuntimeError("No descs.pkl found in MyDrive/algebraic_entropy_runs/. Run your bas

with open(DESCS_PKL, "rb") as f:
    descs = pickle.load(f)

dim = len(descs)
print("[info] Using descs:", DESCS_PKL)
print("[info] dim =", dim)
if dim < 2:
    raise RuntimeError("dim<2: nothing to sweep.")

def j2_to_jj1(j2: int) -> float:
    return 0.0 if int(j2) == 0 else 0.75

def electric_energy_proxy(desc) -> float:
    E = 0.0
    for xs in desc.x:
        E += j2_to_jj1(xs[0]) + j2_to_jj1(xs[1])
    for mu_p in desc.mu:
        for b in mu_p:
            E += j2_to_jj1(b)
    return E
```

```
def flatten_bits(desc):
    bits = []
    for xs in desc.x:
        bits.append(int(xs[0])); bits.append(int(xs[1]))
    for mup in desc.mu:
        for b in mup:
            bits.append(int(b))
    return tuple(bits)

def hamming(a, b):
    return sum((x != y) for x, y in zip(a, b))

def participation_ratio(psi):
    p = np.abs(psi)**2
    return float(1.0 / np.sum(p**2))

bitstrings = [flatten_bits(d) for d in descs]
Lbits = len(bitstrings[0])
if not all(len(b) == Lbits for b in bitstrings):
    raise RuntimeError("Inconsistent bitstring lengths.")
print("[info] bitstring length =", Lbits)

D = np.zeros((dim, dim), dtype=np.int64)
for i in range(dim):
    for j in range(i+1, dim):
        dij = hamming(bitstrings[i], bitstrings[j])
        D[i, j] = dij
        D[j, i] = dij
upper = D[np.triu_indices(dim, k=1)]
print(f"[info] Hamming min/med/max = {int(np.min(upper))}/{float(np.median(upper))}/{int(np

def build_knn_laplacian(D, k_nn=3, alpha_w=0.5):
    n = D.shape[0]
    k = int(max(1, min(k_nn, n-1)))
    A = np.zeros((n, n), dtype=np.float64)
    for i in range(n):
        neigh = [(int(D[i, j]), j) for j in range(n) if j != i]
        neigh.sort(key=lambda t: (t[0], t[1]))
        for d, j in neigh[:k]:
            w = float(np.exp(-alpha_w * d))
            A[i, j] = max(A[i, j], w)
            A[j, i] = max(A[j, i], w)
    deg = np.sum(A, axis=1)
    L = np.diag(deg) - A
    edges = int(np.sum(A > 0) // 2)
    return L, edges, float(np.min(deg)), float(np.max(deg)), k

# Algebraic entropy + CMI
EPS_SVD   = 1e-15
TOL_NORM  = 1e-12
TOL_SFULL = 1e-10
```

```
def alpha_of_desc(desc, i: int, j: int, L: int = 4):
    groups = []
    if i > 1:
        groups.append(desc.x[i-2])
    if j < L:
        groups.append(desc.x[j-1])
    return tuple(groups)

def keyR_of_desc(desc, i: int, j: int):
    mu_R = desc.mu[i-1:j]
    x_internal = desc.x[i-1:(j-1)]
    return (mu_R, x_internal)

def keyC_of_desc(desc, i: int, j: int, L: int = 4):
    if i == 1:
        mu_L = (); x_L = ()
    else:
        mu_L = desc.mu[0:(i-1)]
        x_L  = desc.x[0:(i-2)]
    if j == L:
        mu_R = (); x_R = ()
    else:
        mu_R = desc.mu[j:L]
        x_R  = desc.x[j:(L-1)]
    return (mu_L, x_L, mu_R, x_R)

def S_alg_interval(psi, i, j, eps=EPS_SVD, tol_norm=TOL_NORM):
    trips = defaultdict(list)
    mapR = defaultdict(dict)
    mapC = defaultdict(dict)
    for k, desc in enumerate(descs):
        a  = alpha_of_desc(desc, i, j, 4)
        kR = keyR_of_desc(desc, i, j)
        kC = keyC_of_desc(desc, i, j, 4)
        r = mapR[a].setdefault(kR, len(mapR[a]))
        c = mapC[a].setdefault(kC, len(mapC[a]))
        trips[a].append((r, c, psi[k]))
    S = 0.0
    norm = 0.0
    for a, tc in trips.items():
        dR = len(mapR[a]); dC = len(mapC[a])
        M = np.zeros((dR, dC), dtype=np.complex128)
        for r, c, amp in tc:
            M[r, c] += amp
        s = np.linalg.svd(M, compute_uv=False)
        w = (s.real**2)
        norm += float(np.sum(w[w > 0]))
        w = w[w > eps]
        S -= float(np.sum(w * np.log(w)))
    if abs(norm - 1.0) > tol_norm:
```

```
        raise RuntimeError(f"Schmidt weight check failed: sum s^2 = {norm}")
    return float(S)

# Base H_E
g = 1.0
HE_diag = np.array([(g**2/2.0) * electric_energy_proxy(d) for d in descs], dtype=np.float64)
HE = np.diag(HE_diag).astype(np.complex128)

# Sweep grid
alpha_grid = [1.0, 0.5, 0.2, 0.1, 0.0]
t_grid = [1.0, 3.0, 10.0, 30.0, 100.0]
k_nn = min(3, dim-1)

rows = []
best_by_PR = None
best_by_CMI = None

for alpha_w in alpha_grid:
    L, edges, deg_min, deg_max, k_used = build_knn_laplacian(D, k_nn=k_nn, alpha_w=alpha_w)
    if edges == 0:
        continue
    for t_mix in t_grid:
        H = HE + (t_mix * L).astype(np.complex128)
        evals, evecs = np.linalg.eigh(H)
        psi0 = evecs[:, 0].astype(np.complex128)
        psi0 /= np.linalg.norm(psi0)

        PR = participation_ratio(psi0)
        Delta = float(np.real(evals[1]-evals[0]))
        xi = float(1.0/Delta) if Delta > 0 else None

        S14 = S_alg_interval(psi0, 1, 4)
        if abs(S14) > TOL_SFULL:
            raise RuntimeError("S(full) sanity check failed (unexpected).")

        S12 = S_alg_interval(psi0, 1, 2)
        S23 = S_alg_interval(psi0, 2, 3)
        S2  = S_alg_interval(psi0, 2, 2)
        S13 = S_alg_interval(psi0, 1, 3)
        S24 = S_alg_interval(psi0, 2, 4)

        CMI1 = float(S12 + S23 - S2 - S13)
        CMI2 = float(S13 + S24 - S23 - S14)

        rec = {
            "alpha_w": float(alpha_w),
            "t_mix": float(t_mix),
            "k_nn": int(k_used),
            "edges": int(edges),
            "deg_min": float(deg_min),
            "deg_max": float(deg_max),
```

```
            "E0": float(np.real(evals[0])),
            "Delta": Delta,
            "xi": xi,
            "PR": PR,
            "S12": S12, "S23": S23, "S13": S13, "S24": S24, "S2": S2, "S14": S14,
            "CMI_w1": CMI1,
            "CMI_w2": CMI2,
        }
        rows.append(rec)

        if (best_by_PR is None) or (PR > best_by_PR["PR"]):
            best_by_PR = dict(rec); best_by_PR["_H"]=H; best_by_PR["_evals"]=evals; best_by_

        if PR >= 2.0:
            score = (CMI1 + CMI2)
            if (best_by_CMI is None) or (score < (best_by_CMI["CMI_w1"] + best_by_CMI["CMI_w
                best_by_CMI = dict(rec); best_by_CMI["_H"]=H; best_by_CMI["_evals"]=evals; b

# Save sweep
with open(os.path.join(OUTDIR, "sweep.json"), "w", encoding="utf-8") as f:
    json.dump(rows, f, indent=2)

csv_path = os.path.join(OUTDIR, "sweep.csv")
with open(csv_path, "w", newline="", encoding="utf-8") as f:
    w = csv.DictWriter(f, fieldnames=list(rows[0].keys()))
    w.writeheader()
    w.writerows(rows)

def save_best(tag, best):
    if best is None: return
    np.save(os.path.join(OUTDIR, f"H_{tag}.npy"), best["_H"])
    np.save(os.path.join(OUTDIR, f"evals_{tag}.npy"), best["_evals"])
    np.save(os.path.join(OUTDIR, f"psi0_{tag}.npy"), best["_psi0"])
    tmp = {k: v for k, v in best.items() if not k.startswith("_")}
    with open(os.path.join(OUTDIR, f"best_{tag}.json"), "w", encoding="utf-8") as f:
        json.dump(tmp, f, indent=2)
    print(f"[info] Saved best_{tag}: PR={tmp['PR']}, CMI1={tmp['CMI_w1']}, CMI2={tmp['CMI_w2

save_best("maxPR", best_by_PR)
save_best("minCMI_PRge2", best_by_CMI)

# Make plots
PR = np.array([r["PR"] for r in rows], dtype=float)
Delta = np.array([r["Delta"] for r in rows], dtype=float)
CMI = np.array([r["CMI_w1"] + r["CMI_w2"] for r in rows], dtype=float)
S23 = np.array([r["S23"] for r in rows], dtype=float)
S13 = np.array([r["S13"] for r in rows], dtype=float)

def save_scatter(x, y, xlabel, ylabel, title, path, logy=False):
    plt.figure(figsize=(6,4))
    if logy:
```

```
        plt.yscale("log")
        plt.scatter(x, np.maximum(y, 1e-16), s=30)
    else:
        plt.scatter(x, y, s=30)
    plt.xlabel(xlabel); plt.ylabel(ylabel); plt.title(title)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.savefig(path, dpi=200)
    plt.close()

save_scatter(PR, CMI, "Participation ratio PR", "CMI sum (w=1+w=2)", "PR vs CMI", os.path.jo
save_scatter(Delta, CMI, "Gap ", "CMI sum (w=1+w=2)", " vs CMI", os.path.join(OUTDIR,"Delta_

plt.figure(figsize=(5,5))
plt.scatter(S13, S23, s=30)
plt.xlabel("S(1..3)"); plt.ylabel("S(2..3)")
plt.title("Entropy structure: S(2..3) vs S(1..3)")
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig(os.path.join(OUTDIR,"S23_vs_S13.png"), dpi=200)
plt.close()

# Write paper_summary.txt
def fmt(m):
    return (
        f"alpha_w={m['alpha_w']}, t_mix={m['t_mix']}, k_nn={m['k_nn']}, edges={m['edges']},
        f"Delta={m['Delta']}, xi={m.get('xi', None)}\n"
        f"S12={m['S12']}, S23={m['S23']}, S2={m['S2']}, S13={m['S13']}, S24={m['S24']}\n"
        f"CMI_w1={m['CMI_w1']}, CMI_w2={m['CMI_w2']}, CMI_sum={m['CMI_w1']+m['CMI_w2']}"
    )

lines = []
lines.append(f"Run folder: {OUTDIR}")
lines.append(f"descs.pkl used: {DESCS_PKL}")
lines.append(f"Total configs: {len(rows)}")
lines.append("")
lines.append("Best by PR:")
lines.append(fmt(best_by_PR))
lines.append("")
lines.append("Best by CMI_sum subject to PR>=2:")
lines.append(fmt(best_by_CMI) if best_by_CMI else "None")
lines.append("")
lines.append("Files generated:")
lines.append("  sweep.csv, sweep.json")
lines.append("  PR_vs_CMI.png, Delta_vs_CMI.png, S23_vs_S13.png")
lines.append("  paper_summary.txt")
lines.append("  best_*.json, H_*.npy, evals_*.npy, psi0_*.npy")
lines.append("  overleaf_bundle.zip")

with open(os.path.join(OUTDIR, "paper_summary.txt"), "w", encoding="utf-8") as f:
    f.write("\n".join(lines))
```

```
# Build Overleaf zip
bundle_files = [
    "sweep.csv","sweep.json","paper_summary.txt",
    "PR_vs_CMI.png","Delta_vs_CMI.png","S23_vs_S13.png",
    "best_maxPR.json","best_minCMI_PRge2.json",
    "H_maxPR.npy","evals_maxPR.npy","psi0_maxPR.npy",
]
zip_path = os.path.join(OUTDIR, "overleaf_bundle.zip")
with zipfile.ZipFile(zip_path, "w", compression=zipfile.ZIP_DEFLATED) as z:
    for fname in bundle_files:
        p = os.path.join(OUTDIR, fname)
        if os.path.exists(p):
            z.write(p, arcname=fname)

print("[info] Wrote Overleaf zip:", zip_path)
files.download(zip_path)
```

C Notes on related literature (brief)

The present work is primarily a reproducible toy-model pipeline. For context, entanglement in gauge theories requires care due to constraints and centers (see e.g. [2]). The Kogut–Susskind Hamiltonian formulation provides the standard lattice gauge Hamiltonian setting [1]. In standard quantum information, the condition $I(A : C \mid B) = 0$ characterizes quantum Markov states and admits structural theorems [3].

References

[1] J. Kogut and L. Susskind, "Hamiltonian formulation of Wilson's lattice gauge theories," *Phys. Rev. D* **11** (1975) 395–408.

[2] H. Casini, M. Huerta, and J. A. Rosabal, "Remarks on entanglement entropy for gauge fields," *Phys. Rev. D* **89** (2014) 085012.

[3] P. Hayden, R. Jozsa, D. Petz, and A. Winter, "Structure of states which satisfy strong subadditivity of quantum entropy with equality," *Commun. Math. Phys.* **246** (2004) 359–374.