# Petz recoverability versus Wilson-loop diagnostics in $\mathbb{Z}_2$ lattice gauge theory (2+1D)

Exact diagonalization benchmark on small open lattices

Lluis Eriksson
Independent Researcher
lluiseriksson@gmail.com

January 2026

**Abstract**

We provide a finite-size benchmark testing whether a Petz-type recoverability proxy correlates with Wilson-loop confinement diagnostics in $\mathbb{Z}_2$ lattice gauge theory in 2+1 dimensions. We compute ground states by sparse exact diagonalization on $2 \times 2$ and $2 \times 3$ plaquette lattices with open boundary conditions, using qubits on links and enforcing the gauge-invariant sector via a Gauss-law penalty (verified numerically by $\langle G_v \rangle \simeq 1$ for all vertices). For opposite-corner plaquette patches $A$ and $C$, we define a buffer $B(w)$ by BFS thickening in the link-adjacency graph (with $w = 0$ denoting an empty buffer) and evaluate a trace-renormalized, regularized Petz-type recoverability error $E_{\mathrm{rec}}^{\mathrm{Petz}}(w) = -\log F(\rho_{ABC}, \widehat{\rho}_{ABC}(w))$. As a confinement proxy we use Wilson loops and the Creutz ratio $\chi(2,2)$, treated as an effective string tension $\sigma_{\mathrm{eff}}$. In 2+1 dimensions the natural confinement-length proxy is $\sigma_{\mathrm{eff}}^{-1}$. Across couplings $g$ we observe that larger $\sigma_{\mathrm{eff}}^{-1}$ correlates with larger fixed-collar recovery error $E_{\mathrm{rec}}^{\mathrm{Petz}}(w = 1)$ (small-lattice benchmark).

# Contents

# 1 Introduction

Petz-type recoverability provides an operational measure of approximate quantum Markov structure: given a tripartition $A$–$B$–$C$, one asks how well $\rho_{ABC}$ can be reconstructed from $\rho_{AB}$ using a recovery channel acting only on $B$. In gauge theories, subsystem definitions are subtle due to constraints, and reduced states can depend on the chosen prescription. In companion work we proposed a reproducible protocol for computing Petz-type recoverability on lattices and conjectured links to confinement diagnostics. Here we provide a small-lattice benchmark in a genuine lattice gauge theory.

**Scope.** We work on very small lattices ($2 \times 2$ and $2 \times 3$ plaquettes) where exact diagonalization is feasible. Finite-size and boundary effects are large. Our goal is a falsifiable, reproducible benchmark rather than a thermodynamic-limit statement.

**Dimensional analysis in 2+1 dimensions.** In $d$+1 spacetime dimensions, the string tension has dimension $[\sigma] = (\text{length})^{-(d-1)}$. In 2+1 dimensions ($d = 2$), the natural length scale is therefore $\ell_\sigma \sim \sigma^{-1}$, motivating comparisons to $\sigma_{\text{eff}}^{-1}$.

# 2 Model and setup

## 2.1 $\mathbb{Z}_2$ Hamiltonian lattice gauge theory on links

We work on a spatial $N_x \times N_y$ plaquette lattice with open boundary conditions (OBC), with qubits on links. We use the Hamiltonian

$$H(g) = -g \sum_{\ell} X_\ell \; - \; \frac{1}{g} \sum_{p} \prod_{\ell \in \partial p} Z_\ell \; + \; \Lambda \sum_{v} (I - G_v),$$

where $X_\ell$ and $Z_\ell$ are Pauli operators on link $\ell$, the plaquette term is the product around $\partial p$, and the gauge generators are

$$G_v := \prod_{\ell \ni v} X_\ell.$$

We take $\Lambda = 50$ in the numerics and verify $\langle G_v \rangle \simeq 1$ for all vertices in the computed ground state.

## 2.2 Tripartition geometry (PARCHES) and collaring rule

Subsystems are defined as subsets of links (an extended-Hilbert-space tensor-product of link degrees of freedom). We choose $A$ and $C$ as *plaquette patches* (four links) in opposite corners: $A$ is the southwest plaquette patch and $C$ is the northeast plaquette patch.

We define the collar $B(w)$ by BFS thickening in the link-adjacency graph (two links adjacent if they share a vertex):

$$B(0) = \emptyset, \qquad B(w) = \{\ell : \; \text{dist}_{\text{link}}(\ell, A) \leq w\} \setminus (A \cup C), \quad w \geq 1.$$

*Remark* 2.1. On very small lattices, $E_{\text{rec}}^{\text{Petz}}(w)$ need not be monotone in $w$ due to finite-size saturation and the trace normalization used for the regularized Petz-type reconstruction. We therefore emphasize fixed-collar comparisons (notably $w = 1$) and robustness checks, rather than interpreting $E_{\text{rec}}^{\text{Petz}}(w)$ as a clean exponential decay profile at these sizes.

# 3 Observables

## 3.1 Regularized Petz-type recoverability error

For each collar width $w$, we compute reduced states $\rho_{ABC}$, $\rho_{AB}$, $\rho_B$, and $\rho_{BC}$. We use a regularized Petz-type map

$$\mathcal{R}^{\mathrm{Petz}(\delta)}_{B\to BC}(X_B) = \rho_{BC}^{1/2}\Big(\rho_{B,\delta}^{-1/2} X_B \rho_{B,\delta}^{-1/2} \otimes I_C\Big)\rho_{BC}^{1/2}, \qquad \rho_{B,\delta} := \rho_B + \delta I,$$

with $\delta = 10^{-12}$, and normalize the reconstructed operator by its trace before computing fidelity. We report

$$E^{\mathrm{Petz}}_{\mathrm{rec}}(w) := -\log F(\rho_{ABC}, \widehat{\rho}_{ABC}(w)).$$

## 3.2 Wilson loops and Creutz proxy

For a closed loop $\gamma$ we define the $\mathbb{Z}_2$ Wilson loop operator $W(\gamma) = \prod_{\ell\in\gamma} Z_\ell$. As a practical proxy for string tension on tiny lattices, we compute the Creutz ratio

$$\chi(2,2) = -\log \frac{\langle W(2,2)\rangle\langle W(1,1)\rangle}{\langle W(2,1)\rangle\langle W(1,2)\rangle},$$

and set $\sigma_{\mathrm{eff}} := \chi(2,2)$.

*Remark* 3.1 (Dimensional analysis in 2+1 dimensions). In 2+1 spacetime dimensions, $[\sigma] = (\mathrm{length})^{-1}$, so the natural confinement-length proxy is $\sigma_{\mathrm{eff}}^{-1}$ (not $\sigma_{\mathrm{eff}}^{-1/2}$).

# 4 Results

## 4.1 Summary table

| $N_x \times N_y$ | $g$ | $E_0$ | $\sigma_{\mathrm{eff}}$ | $1/\sigma_{\mathrm{eff}}$ | $E^{\mathrm{Petz}}_{\mathrm{rec}}(w{=}0)$ | $E^{\mathrm{Petz}}_{\mathrm{rec}}(w{=}1)$ |
|---|---|---|---|---|---|---|
| $2\times 2$ | 0.5 | $-9.5660$ | 0.150780 | 6.632169 | $1.566201\times 10^{-1}$ | $3.685747\times 10^{-1}$ |
| $2\times 2$ | 1.0 | $-12.4973$ | 1.073026 | 0.931944 | $1.202218\times 10^{-3}$ | $4.432791\times 10^{-3}$ |
| $2\times 2$ | 2.0 | $-24.0625$ | 2.418666 | 0.413451 | $4.281872\times 10^{-6}$ | $1.848438\times 10^{-5}$ |
| $2\times 3$ | 0.5 | $-13.9474$ | 0.102507 | 9.755459 | $4.304489\times 10^{-1}$ | $4.261377\times 10^{-1}$ |
| $2\times 3$ | 1.0 | $-17.7472$ | 1.059645 | 0.943713 | $4.315631\times 10^{-2}$ | $4.311739\times 10^{-2}$ |
| $2\times 3$ | 2.0 | $-34.0937$ | 2.417752 | 0.413607 | $2.869337\times 10^{-3}$ | $2.867963\times 10^{-3}$ |

Table 1: Finite-size benchmark for $\mathbb{Z}_2$ lattice gauge theory (OBC, link qubits). The ground state lies in the gauge-invariant sector (numerically verified by $\langle G_v\rangle \simeq 1$ for all vertices). We report $\sigma_{\mathrm{eff}} := \chi(2,2)$ as a Creutz proxy and compare fixed-collar recoverability $E^{\mathrm{Petz}}_{\mathrm{rec}}(w=1)$ against the natural 2+1D length proxy $1/\sigma_{\mathrm{eff}}$.

## 4.2 Recoverability profiles

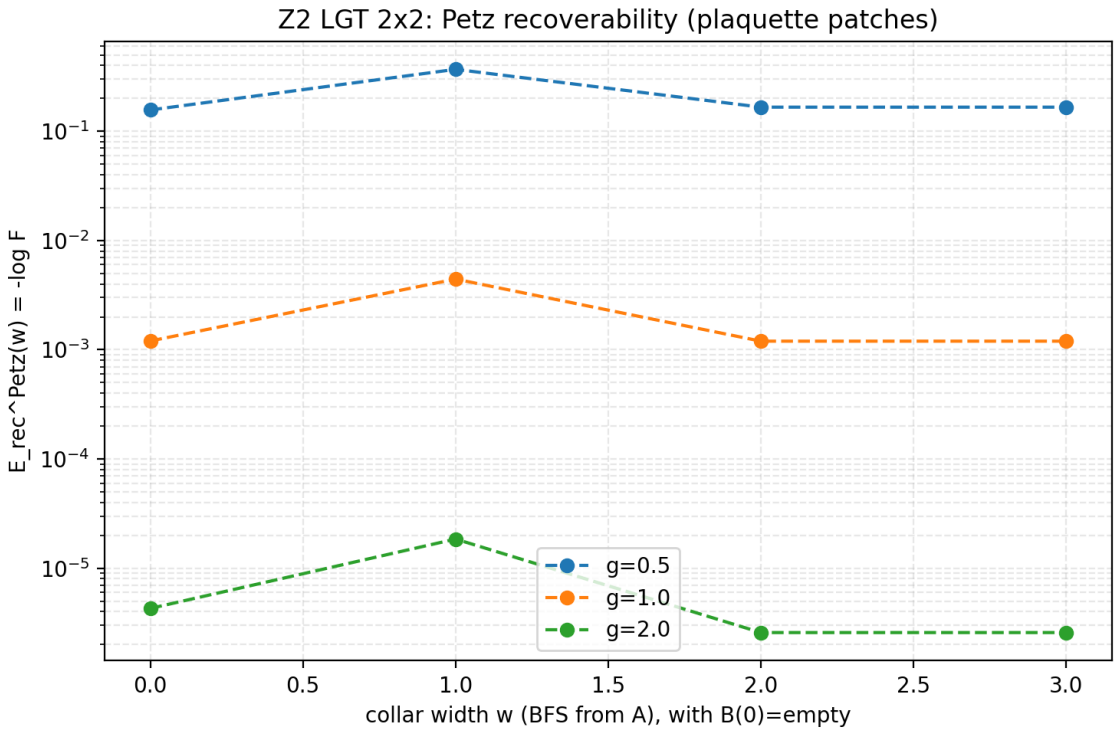


Figure 1: Petz-type recoverability profile $E_{\mathrm{rec}}^{\mathrm{Petz}}(w)$ on the $2 \times 2$ plaquette lattice (12 links) for plaquette-patch tripartitions with opposite-corner patches $A$ and $C$. The collar $B(w)$ is defined by BFS thickening in the link-adjacency graph, with $B(0) = \emptyset$.

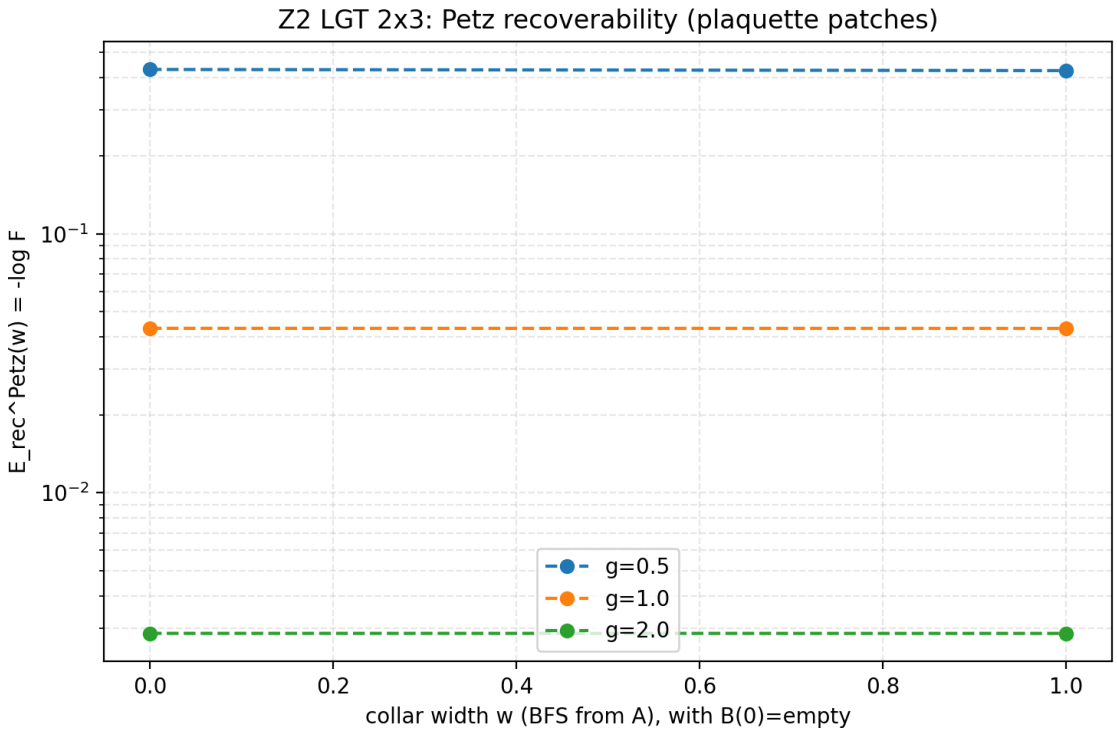


Figure 2: Petz-type recoverability profile $E_{\mathrm{rec}}^{\mathrm{Petz}}(w)$ on the $2 \times 3$ plaquette lattice (17 links) for the same plaquette-patch geometry. Dense Petz reconstruction restricts feasible collar widths; here we report $w = 0, 1$.

### 4.3 Fixed-collar recoverability versus confinement-length proxy

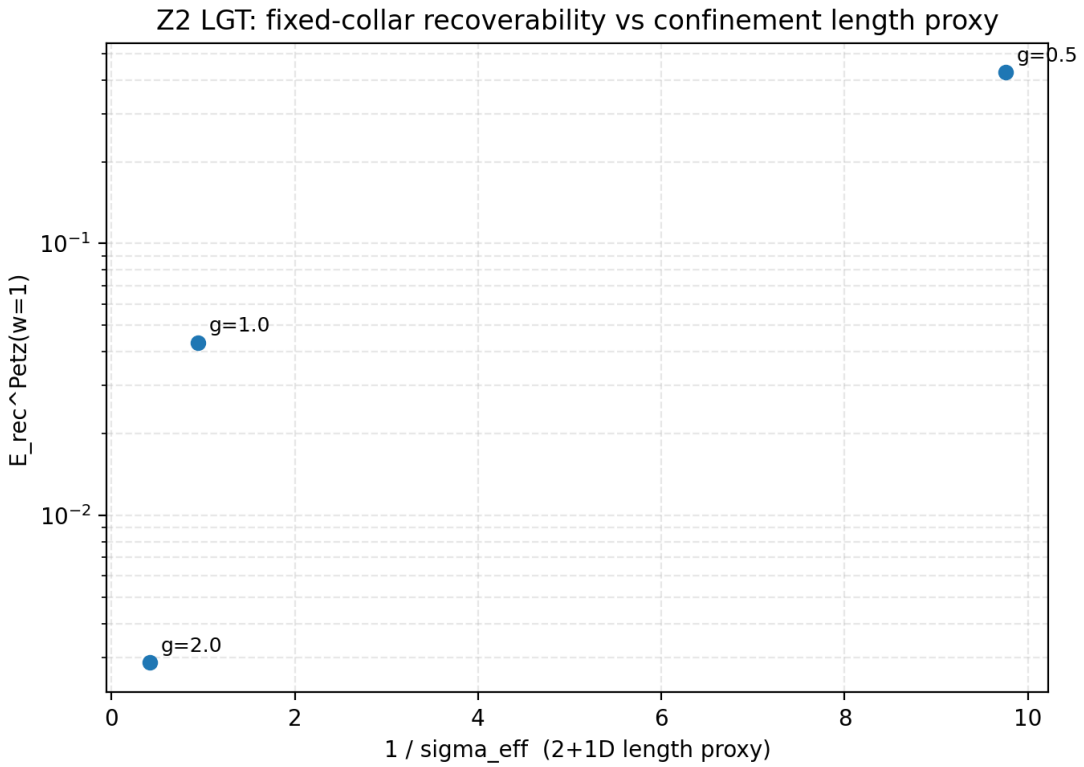


Figure 3: Fixed-collar recoverability versus confinement-length proxy on the $2 \times 3$ plaquette lattice. The horizontal axis is $\sigma_{\text{eff}}^{-1}$ with $\sigma_{\text{eff}} = \chi(2,2)$ (Creutz ratio), and the vertical axis is the fixed-collar recovery error $E_{\text{rec}}^{\text{Petz}}(w=1)$. Across couplings $g$, larger $\sigma_{\text{eff}}^{-1}$ correlates with larger $E_{\text{rec}}^{\text{Petz}}(w=1)$ (finite-size benchmark).

## 5 Discussion

On $2 \times 3$, the near-equality of $E_{\text{rec}}^{\text{Petz}}(w=0)$ and $E_{\text{rec}}^{\text{Petz}}(w=1)$ indicates rapid finite-size saturation of the buffer effect for this geometry. On $2 \times 2$, mild non-monotonicity in $w$ can occur; we therefore emphasize fixed-collar comparisons and treat collar dependence as a diagnostic at these sizes.

The primary observation is a clear correlation between fixed-collar recoverability $E_{\text{rec}}^{\text{Petz}}(w=1)$ and the confinement-length proxy $\sigma_{\text{eff}}^{-1}$. Extending to larger collars and lattices will require tensor-network contractions or alternative recovery implementations.

## A Reproducibility: code to regenerate the figures

The figures in this paper can be regenerated by running the Python script below. It computes ground states by sparse exact diagonalization, checks the gauge constraint $\langle G_v \rangle \simeq 1$ in the resulting state, evaluates the Petz-type recoverability profile for the plaquette-patch geometry, and produces the PNG plots used in the main text.

**Dependencies.** Python 3 with `numpy`, `scipy`, and `matplotlib`.

**Run.** Save the listing to `paper_d_reproduce.py` and run:

```
python paper_d_reproduce.py
```

This generates: `z2_2x2_recoverability_v2.png`, `z2_2x3_recoverability_v2.png`, `z2_2x3_fixed_w` and `paper_d_results.csv`.

Listing 1: paper_d_reproduce.py (reproducibility script)

```
# NOTE: This is the same code used to generate the CSV/PNGs for Paper D v1.
import csv
import numpy as np
import scipy.sparse as sp
import scipy.sparse.linalg as spla
import scipy.linalg as la
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt

def idx_h(x, y, Nx, Ny): return y * Nx + x
def idx_v(x, y, Nx, Ny):
    Nh = Nx * (Ny + 1)
    return Nh + y * (Nx + 1) + x

def build_lattice(Nx, Ny):
    def vid(x, y): return y * (Nx + 1) + x
    link_endpoints = []
    for y in range(Ny + 1):
        for x in range(Nx):
            link_endpoints.append((vid(x, y), vid(x + 1, y)))
    for y in range(Ny):
        for x in range(Nx + 1):
            link_endpoints.append((vid(x, y), vid(x, y + 1)))
    Nlinks = len(link_endpoints)

    plaquettes = []
    for y in range(Ny):
        for x in range(Nx):
            pl = [idx_h(x,y,Nx,Ny), idx_v(x+1,y,Nx,Ny), idx_h(x,y+1,Nx,Ny), idx_v(x,y,
                Nx,Ny)]
            plaquettes.append(pl)

    Nv = (Nx + 1) * (Ny + 1)
    stars = [[] for _ in range(Nv)]
    for ell, (a, b) in enumerate(link_endpoints):
        stars[a].append(ell); stars[b].append(ell)

    v2links = [[] for _ in range(Nv)]
    for ell, (a, b) in enumerate(link_endpoints):
        v2links[a].append(ell); v2links[b].append(ell)
    adjacency = [set() for _ in range(Nlinks)]
    for v in range(Nv):
        Ls = v2links[v]
        for i in Ls:
            for j in Ls:
                if i != j:
                    adjacency[i].add(j)
```

```
48
49      return Nlinks, plaquettes, stars, adjacency
50
51  def mask_from_links(link_list):
52      m = 0
53      for ell in link_list:
54          m |= (1 << int(ell))
55      return m
56
57  def parity_u64(x): return x.bit_count() & 1
58
59  def build_hamiltonian_z2(Nlinks, plaquette_masks, star_masks, g, Lambda=0.0):
60      dim = 1 << Nlinks
61      states = np.arange(dim, dtype=np.uint32 if Nlinks <= 32 else np.uint64)
62      invg = 1.0 / float(g)
63      diag = np.zeros(dim, dtype=np.float64)
64
65      for mp in plaquette_masks:
66          par = np.fromiter((parity_u64(int(s) & mp) for s in states), count=dim, dtype
                =np.int8)
67          eig = 1.0 - 2.0 * par
68          diag += -(invg) * eig
69
70      if Lambda != 0.0:
71          diag += float(Lambda) * len(star_masks)
72
73      rows = []; cols = []; data = []
74
75      rows.extend(range(dim)); cols.extend(range(dim)); data.extend(diag.tolist())
76
77      for ell in range(Nlinks):
78          mask = 1 << ell
79          tgt = states ^ mask
80          rows.extend(states.tolist()); cols.extend(tgt.tolist())
81          data.extend(((-float(g)) * np.ones(dim, dtype=np.float64)).tolist())
82
83      if Lambda != 0.0:
84          for mv in star_masks:
85              tgt = states ^ mv
86              rows.extend(states.tolist()); cols.extend(tgt.tolist())
87              data.extend(((-float(Lambda)) * np.ones(dim, dtype=np.float64)).tolist())
88
89      H = sp.coo_matrix((np.array(data, dtype=np.float64),
90                         (np.array(rows), np.array(cols))), shape=(dim, dim)).tocsr()
91      H.sum_duplicates()
92      return H
93
94  def ground_state(H, tol=1e-10, maxiter=30000):
95      vals, vecs = spla.eigsh(H, k=1, which="SA", tol=tol, maxiter=maxiter)
96      v = vecs[:, 0].astype(np.complex128)
97      v /= np.linalg.norm(v)
98      return float(vals[0]), v
99
100 def plaquette_patch_A_C(Nx, Ny):
101     A = [idx_h(0,0,Nx,Ny), idx_v(1,0,Nx,Ny), idx_h(0,1,Nx,Ny), idx_v(0,0,Nx,Ny)]
102     C = [idx_h(Nx-1,Ny-1,Nx,Ny), idx_v(Nx,Ny-1,Nx,Ny), idx_h(Nx-1,Ny,Nx,Ny), idx_v(
            Nx-1,Ny-1,Nx,Ny)]
103     return sorted(set(A)), sorted(set(C))
104
```

```
def bfs_distances(adjacency, sources):
   n = len(adjacency)
   dist = np.full(n, fill_value=-1, dtype=np.int32)
   q = []
   for s in sources:
      dist[s] = 0; q.append(s)
   head = 0
   while head < len(q):
      u = q[head]; head += 1
      for v in adjacency[u]:
         if dist[v] < 0:
            dist[v] = dist[u] + 1
            q.append(v)
   return dist

def buffer_Bw(adjacency, A, C, w):
   if w <= 0:
      return []
   distA = bfs_distances(adjacency, A)
   B = [i for i, d in enumerate(distA) if (d >= 0 and d <= w and i not in A and i
       not in C)]
   return sorted(B)

def reduce_pure_to_density_qubits(psi, N, keep_qubits):
   keep = list(keep_qubits)
   trace = [i for i in range(N) if i not in keep]
   psi_t = psi.reshape([2]*N)
   perm = keep + trace
   psi_perm = np.transpose(psi_t, axes=perm)
   d_keep = 1 << len(keep)
   d_tr = 1 << len(trace)
   M = psi_perm.reshape(d_keep, d_tr)
   rho = M @ M.conj().T
   return 0.5*(rho + rho.conj().T)

def partial_trace_rho(rho, dims, keep):
   dims = list(dims)
   N = len(dims)
   keep = list(keep)
   trace = [i for i in range(N) if i not in keep]
   d_keep = int(np.prod([dims[i] for i in keep]))
   d_tr = int(np.prod([dims[i] for i in trace]))
   rho = np.asarray(rho, dtype=np.complex128)
   rho_t = rho.reshape(dims + dims)
   perm = keep + trace + [i+N for i in keep] + [i+N for i in trace]
   rho_p = np.transpose(rho_t, axes=perm).reshape(d_keep, d_tr, d_keep, d_tr)
   out = np.trace(rho_p, axis1=1, axis2=3)
   return 0.5*(out + out.conj().T)

def sqrtm_psd(M, eps=0.0):
   w, V = la.eigh(np.asarray(M))
   w = np.maximum(w, eps)
   return (V * np.sqrt(w)) @ V.conj().T

def invsqrtm_pd(M):
   w, V = la.eigh(np.asarray(M))
   if np.min(w) <= 0:
      raise ValueError("Matrix not strictly positive; increase delta.")
   return (V * (1.0/np.sqrt(w))) @ V.conj().T
```

```
163
164 def fidelity_squared(rho, sigma):
165     sr = sqrtm_psd(rho)
166     inner = sr @ sigma @ sr
167     return (np.trace(sqrtm_psd(inner)).real)**2
168
169 def petz_recoverability_from_rhoABC(rho_ABC, dA, dB, dC, delta=1e-12):
170     dims = [dA, dB, dC]
171     rho_AB = partial_trace_rho(rho_ABC, dims, keep=[0,1])
172     rho_B = partial_trace_rho(rho_ABC, dims, keep=[1])
173     rho_BC = partial_trace_rho(rho_ABC, dims, keep=[1,2])
174     rho_B_delta = rho_B + delta*np.eye(dB, dtype=np.complex128)
175     O_B = invsqrtm_pd(rho_B_delta)
176     sqrt_rho_BC = sqrtm_psd(rho_BC)
177     IA = np.eye(dA, dtype=np.complex128)
178     IC = np.eye(dC, dtype=np.complex128)
179     sandwich = np.kron(IA, O_B)
180     middle = sandwich @ rho_AB @ sandwich
181     middle = 0.5*(middle + middle.conj().T)
182     left = np.kron(IA, sqrt_rho_BC)
183     sigma = left @ np.kron(middle, IC) @ left
184     sigma = 0.5*(sigma + sigma.conj().T)
185     tr_sigma = float(np.trace(sigma).real)
186     sigmaN = sigma / tr_sigma
187     F = float(np.clip(fidelity_squared(rho_ABC, sigmaN), 0.0, 1.0))
188     E = -np.log(F + 1e-16)
189     return E, F, tr_sigma
190
191 def expval_Z_string(psi, mask, Nlinks):
192     dim = 1 << Nlinks
193     states = np.arange(dim, dtype=np.uint32 if Nlinks <= 32 else np.uint64)
194     par = np.fromiter((parity_u64(int(s) & mask) for s in states), count=dim, dtype=
        np.int8)
195     eig = 1.0 - 2.0 * par
196     prob = (psi.conj() * psi).real
197     return float(np.sum(prob * eig))
198
199 def expval_X_string(psi, mask, Nlinks):
200     dim = 1 << Nlinks
201     s = np.arange(dim, dtype=np.uint32 if Nlinks <= 32 else np.uint64)
202     t = s ^ mask
203     return float(np.vdot(psi, psi[t]).real)
204
205 def gauge_diagnostics(psi, star_masks, Nlinks):
206     vals = [expval_X_string(psi, mv, Nlinks) for mv in star_masks]
207     return float(np.min(vals)), float(np.mean(vals))
208
209 def rectangle_loop_mask(Nx, Ny, x0, y0, R, T):
210     links = set()
211     for x in range(x0, x0+R):
212         links.add(idx_h(x, y0, Nx, Ny))
213         links.add(idx_h(x, y0+T, Nx, Ny))
214     for y in range(y0, y0+T):
215         links.add(idx_v(x0, y, Nx, Ny))
216         links.add(idx_v(x0+R, y, Nx, Ny))
217     return mask_from_links(sorted(links))
218
219 def creutz_ratio(W_RT, W_Rm1Tm1, W_Rm1T, W_RTm1):
220     eps = 1e-16
```

```
    return -np.log((W_RT*W_Rm1Tm1 + eps) / (W_Rm1T*W_RTm1 + eps))

def run_lattice(Nx, Ny, g_list, Lambda, delta, w_list, max_abc_qubits, out_prefix):
    Nlinks, plaquettes, stars, adjacency = build_lattice(Nx, Ny)
    plaquette_masks = [mask_from_links(pl) for pl in plaquettes]
    star_masks = [mask_from_links(st) for st in stars if len(st) > 0]

    A, C = plaquette_patch_A_C(Nx, Ny)
    Bw_dict = {w: buffer_Bw(adjacency, A, C, w) for w in w_list}

    rows = []
    per_g = []

    for g in g_list:
        H = build_hamiltonian_z2(Nlinks, plaquette_masks, star_masks, g=g, Lambda=
            Lambda)
        E0, psi = ground_state(H)
        mnG, avG = gauge_diagnostics(psi, star_masks, Nlinks)

        # W(1,1) avg
        W11_vals = []
        for y0 in range(Ny):
            for x0 in range(Nx):
                W11_vals.append(expval_Z_string(psi, rectangle_loop_mask(Nx, Ny, x0, y0,
                    1, 1), Nlinks))
        W11_avg = float(np.mean(W11_vals))

        # sigma_eff = chi(2,2) at (0,0) (simple benchmark)
        sigma_eff = np.nan
        inv_sigma = np.nan
        if Nx >= 2 and Ny >= 2:
            W22 = expval_Z_string(psi, rectangle_loop_mask(Nx, Ny, 0, 0, 2, 2), Nlinks
                )
            W21 = expval_Z_string(psi, rectangle_loop_mask(Nx, Ny, 0, 0, 2, 1), Nlinks
                )
            W12 = expval_Z_string(psi, rectangle_loop_mask(Nx, Ny, 0, 0, 1, 2), Nlinks
                )
            W11 = expval_Z_string(psi, rectangle_loop_mask(Nx, Ny, 0, 0, 1, 1), Nlinks
                )
            sigma_eff = creutz_ratio(W22, W11, W21, W12)
            inv_sigma = 1.0 / sigma_eff if sigma_eff != 0 else np.nan

        Erec_dict = {}
        for w in w_list:
            B = Bw_dict[w]
            keep = A + B + C
            k = len(keep)

            if k > max_abc_qubits:
                rows.append([Nx,Ny,g,E0,mnG,avG,w,k,np.nan,np.nan,np.nan,W11_avg,
                    sigma_eff,inv_sigma])
                continue

            rho_ABC = reduce_pure_to_density_qubits(psi, Nlinks, keep)
            dA = 1 << len(A)
            dB = 1 << len(B)
            dC = 1 << len(C)

            Erec, F, tr_sigma = petz_recoverability_from_rhoABC(rho_ABC, dA, dB, dC,
```

```
                delta=delta)
            Erec_dict[w] = Erec
            rows.append([Nx,Ny,g,E0,mnG,avG,w,k,Erec,F,tr_sigma,W11_avg,sigma_eff,
                inv_sigma])

        per_g.append((g, Erec_dict, sigma_eff, inv_sigma))

    # plot recoverability
    plt.figure(figsize=(7.5,5))
    for (g, Erec_dict, _, _) in per_g:
        ws = sorted(Erec_dict.keys())
        Es = [max(Erec_dict[w], 1e-16) for w in ws]
        plt.plot(ws, Es, "o--", label=f"g={g}")
    plt.yscale("log")
    plt.xlabel("collar width w (BFS from A), with B(0)=empty")
    plt.ylabel("E_rec^Petz(w) = -log F")
    plt.title(f"Z2 LGT {Nx}x{Ny}: Petz recoverability (plaquette patches)")
    plt.grid(True, which="both", ls="--", alpha=0.3)
    plt.legend()
    plt.tight_layout()
    plt.savefig(f"{out_prefix}_recoverability_v2.png", dpi=200)

    return rows

def main():
    g_list = [0.5, 1.0, 2.0]
    Lambda = 50.0
    delta = 1e-12
    w_list = [0,1,2,3]
    max_abc_qubits = 13

    rows = []
    rows += run_lattice(2,2,g_list,Lambda,delta,w_list,max_abc_qubits,"z2_2x2")
    rows += run_lattice(2,3,g_list,Lambda,delta,w_list,max_abc_qubits,"z2_2x3")

    header = ["Nx","Ny","g","E0","min_Gv","mean_Gv","w","ABC_qubits","E_recP","F","
        tr_sigma","W11_avg","sigma_eff","inv_sigma_eff"]
    with open("paper_d_results.csv","w",newline="",encoding="utf-8") as f:
        wr = csv.writer(f); wr.writerow(header); wr.writerows(rows)

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

## References


[1] J. B. Kogut, An introduction to lattice gauge theory and spin systems, *Reviews of Modern Physics* **51**, 659–713 (1979).

[2] D. Petz, Sufficient subalgebras and the relative entropy of states of a von Neumann algebra, *Communications in Mathematical Physics* **105**, 123–131 (1986).

[3] O. Fawzi and R. Renner, Quantum conditional mutual information and approximate Markov chains, *Communications in Mathematical Physics* **340**, 575–611 (2015).

[4] W. Donnelly, Decomposition of entanglement entropy in lattice gauge theory, *Physical Review D* **85**, 085004 (2012).

[5] H. Casini, M. Huerta, and J. A. Rosabal, Remarks on entanglement entropy for gauge fields, *Physical Review D* **89**, 085012 (2014).