Petz recoverability versus Wilson-loop diagnostics in 2+1D $\mathbb{Z}_2$ lattice gauge theory:
benchmarks by exact diagonalization and tensor-network ladders

Lluis Eriksson

January 2026

Abstract

We provide reproducible finite-size benchmarks 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 first present an exact-diagonalization (ED) benchmark on 2×2 and 2×3 plaquette lattices with open boundary conditions and a Gauss-law penalty term, verifying numerically that $\langle G_v \rangle \sim 1$ for all vertices. We evaluate a trace-renormalized, regularized Petz-type recoverability error $E_{\rm rec}^{\rm Petz}(w) = -\log F(\rho_{ABC}, \hat{\rho}_{ABC}(w))$ for separated subsystems and compare it to confinement proxies based on Wilson loops and Creutz ratios, interpreted as an effective string tension $\sigma_{\rm eff}$. We then extend to larger systems using TeNPy DMRG on ladder geometries $2 \times L$ and report multi-L trends as well as a bond-dimension stability check; in the ladder TN part we report $E_{\rm rec}^{\rm Petz}$ as a function of contiguous buffer size $|B|$ in MPS site ordering (proxy), rather than the BFS collar width w used in the ED part. All plots and CSV data in this paper can be regenerated by running the scripts provided in the appendix.

1 Overview and claims (scope)

This work is a *benchmark* study. We do not claim thermodynamic-limit scaling exponents. The goals are:

- (ED) demonstrate end-to-end reproducibility for a gauge-theory instance with strict Gauss checks on small lattices;
- (TN ladders) test whether qualitative trends persist on larger ladders $2 \times L$ beyond ED;
- provide scripts and fixed outputs (CSV/PNG) suitable for community verification.

2 Model and diagnostics

We consider $\mathbb{Z}_2$lattice gauge theory with qubits on links and open boundary conditions [1]. The Hamiltonian used in our benchmarks has the form

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

where $\Lambda > 0$ energetically penalizes gauge-violating sectors. We validate gauge invariance a posteriori by reporting $\min_v \langle G_v \rangle$ and $\mathrm{mean}_v \langle G_v \rangle$.

Component	System sizes	Key numerical settings
Exact diagonalization (ED)	2×2, 2×3 plaquettes	sparse ED; Gauss penalty Λ (see script)
Tensor-network ladders (TN)	ladders $2 \times L$, $L \in \{4, 6, 8\}$	TeNPy DMRG; $\chi_{\max} = 96$; $g \in \{0.5, 0.8, 1.0, 1.2, 1.5, 2.0\}$

Table 1: Benchmark overview (reproducible by the scripts in the appendix).

Confinement proxy. We measure Wilson loops and compute Creutz ratios $\chi(2, 2)$ where defined, and treat

$$\sigma_{\text{eff}} := \overline{\chi(2, 2)}, \tag{2}$$

where the overline denotes the average over all placements on the given finite lattice where the loop fits. In 2+1 dimensions, the associated confinement-length proxy is $1/\sigma_{\text{eff}}$.

Recoverability proxy. Given a tripartition A–$B(w)$–C, we compute a Petz-type recoverability error [2, 3]

$$E_{\text{rec}}^{\text{Petz}}(w) = -\log F(\rho_{ABC}, \hat{\rho}_{ABC}(w)), \tag{3}$$

with $\hat{\rho}_{ABC}(w)$ produced by a regularized Petz recovery map and with trace-renormalization as implemented in the accompanying code.

2.1 Benchmark summary (sizes and parameters)

Table 1 summarizes the system sizes and the main numerical parameters used in the ED and TN benchmarks.

3 Exact diagonalization benchmark on 2×2 and 2×3

3.1 Reproducibility summary

The ED script produces a CSV and three figures:

- `paper_d_results.csv`
- `z2_2x2_recoverability_v2.png`
- `z2_2x3_recoverability_v2.png`
- `z2_2x3_fixed_w1_vs_invsigma_v2.png`

3.2 Figures (ED)

4 Tensor-network ladders $2 \times L$ (TeNPy DMRG)

4.1 Geometry note (important)

In the tensor-network ladders benchmark we use a computationally convenient proxy tripartition in which A, B, and C are chosen as *contiguous segments in the MPS site ordering*. Therefore, $|B|$ should be interpreted as a *contiguous buffer size in MPS ordering*, not as a geometric BFS collar in the link-adjacency graph. This choice is explicit in the script.

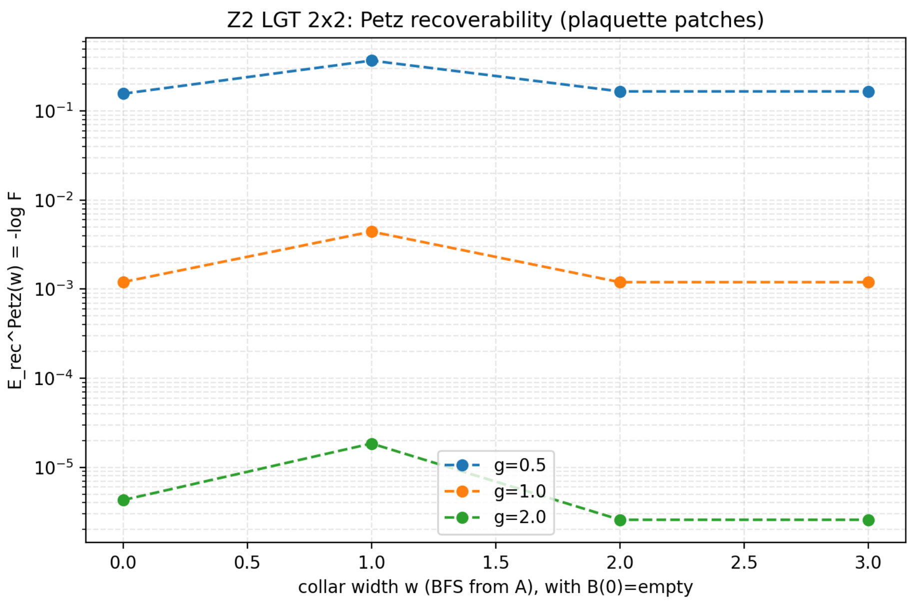


Figure 1: ED benchmark on 2×2: $E_{\mathrm{rec}}^{\mathrm{Petz}}(w)$ versus buffer parameter w (as defined in the ED script).

$\chi_{\max}$	E_0 (with shift)	σ_{eff}	$E_{\mathrm{rec}}^{\mathrm{Petz}}(\|B\| = 1)$	$E_{\mathrm{rec}}^{\mathrm{Petz}}(\|B\| = 2)$
96	-43.851122487503	2.497241	1.692920×10^{-4}	5.840812×10^{-4}
192	-43.851122487498	2.497241	1.692840×10^{-4}	5.840946×10^{-4}

Table 2: Warm-start bond-dimension stability check at $(L, g) = (8, 1.0)$ (see `paper_e_convergence_check_warmstart.py` and `paper_e_convergence_check_warmstart.csv`).

4.2 Figures (TN)

4.3 DMRG bond-dimension stability check

We include a warm-start bond-dimension stability check at $(L, g) = (8, 1.0)$, continuing the same MPS from $\chi_{\max} = 96$ to $\chi_{\max} = 192$. The script `paper_e_convergence_check_warmstart.py` writes `paper_e_convergence_check_warmstart.csv`.

A representative warm-start convergence check at $(L, g) = (8, 1.0)$ is shown in Table 2. Here the MPS obtained at $\chi_{\max} = 96$ is continued at $\chi_{\max} = 192$ (same state branch), and the observables are stable to all shown digits.

5 Reproducibility

All CSVs/PNGs in this paper are generated by the scripts included in the appendix. A minimal validation script for the ladder results is also provided.

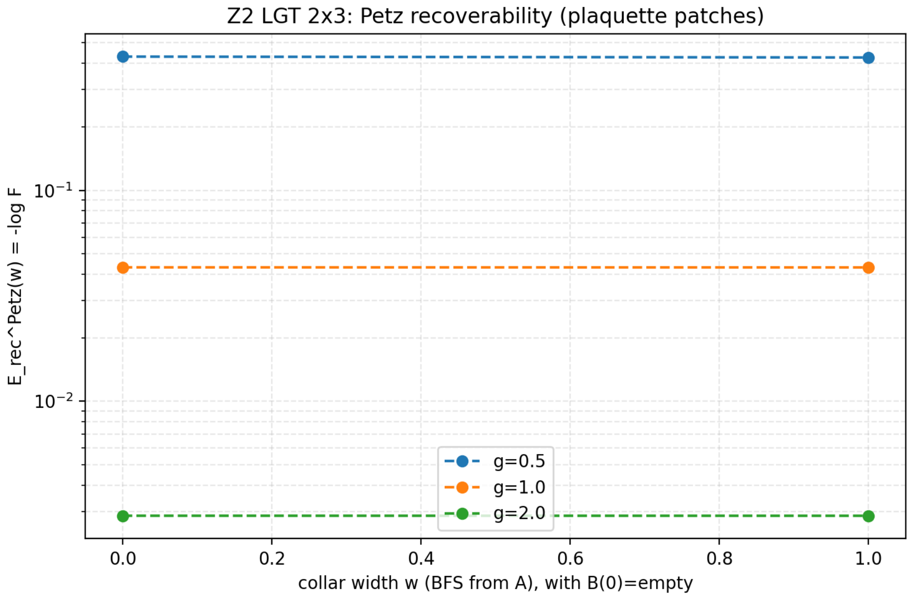


Figure 2: ED benchmark on 2×3: $E_{\mathrm{rec}}^{\mathrm{Petz}}(w)$ versus buffer parameter w (as defined in the ED script).

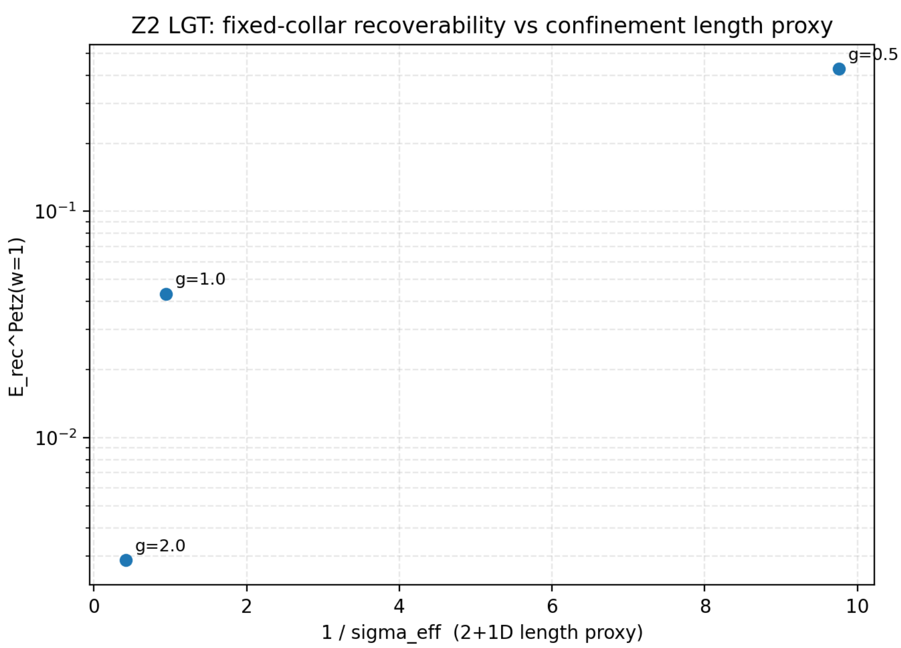


Figure 3: ED "money plot": fixed-collar recoverability versus confinement proxy $1/\sigma_{\mathrm{eff}}$.

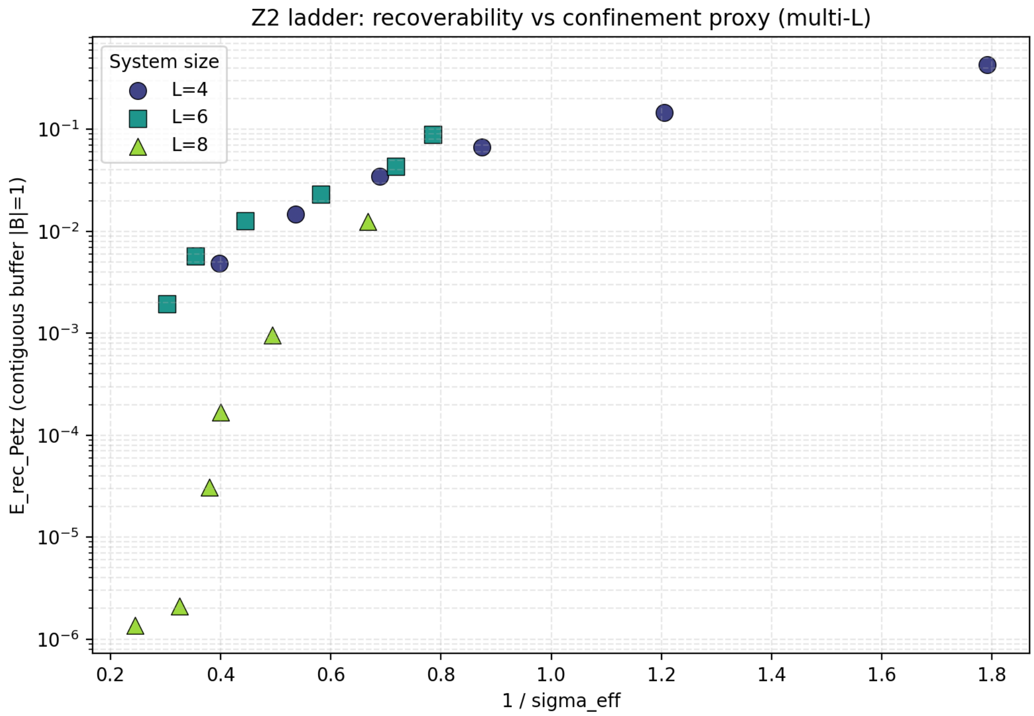


Figure 4: Tensor-network ladders: fixed contiguous buffer ($|B| = 1$) in the MPS site ordering (proxy) recoverability versus $1/\sigma_{\text{eff}}$, colored by L.

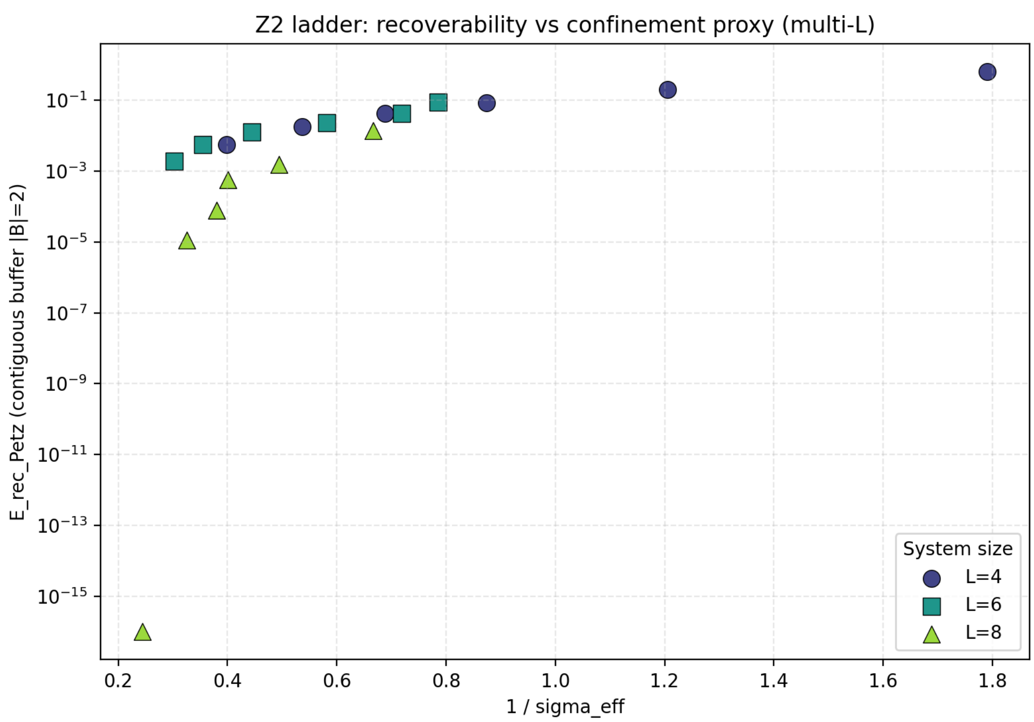
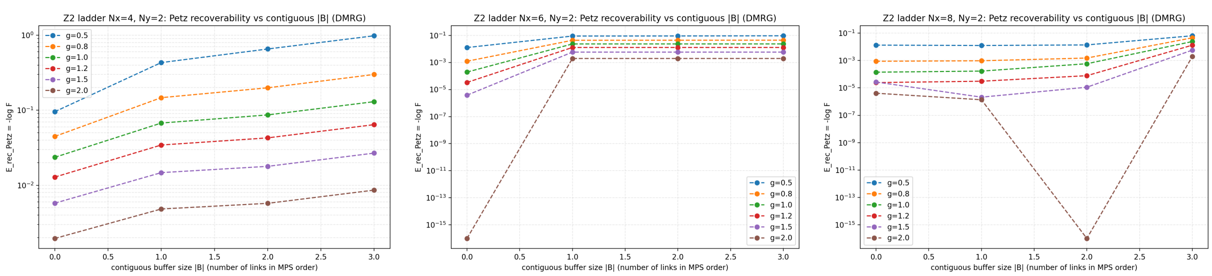


Figure 5: Tensor-network ladders: fixed contiguous buffer ($|B| = 2$) in the MPS site ordering (proxy) recoverability versus $1/\sigma_{\text{eff}}$, colored by L.



(a) $L = 4$ (b) $L = 6$ (c) $L = 8$

Figure 6: Recoverability profiles $E_{\text{rec}}^{\text{Petz}}(|B|)$ on ladder geometries $2 \times L$ for representative L.

References

[1] J. B. Kogut, An introduction to lattice gauge theory and spin systems, *Rev. Mod. Phys.* **51**, 659–713 (1979).

[2] D. Petz, Sufficient subalgebras and the relative entropy of states of a von Neumann algebra, *Commun. Math. Phys.* **105**, 123–131 (1986).

[3] O. Fawzi and R. Renner, Quantum conditional mutual information and approximate Markov chains, *Commun. Math. Phys.* **340**, 575–611 (2015).

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

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

A Reproduction scripts

A.1 Exact diagonalization reproduction script (paper_d_reproduce.py)

Missing file: paper_d_reproduce.py

A.2 Tensor-network ladders reproduction script (paper_e_reproduce.py)

```
# paper_e_reproduce.py
# Paper E v1.1: Multi-L scan on Z2 LGT ladders (Ny=2 plaquettes) with
   TeNPy+DMRG and Gauss penalty.
#
# Outputs (in current folder):
#   - paper_e_results.csv
#   - z2_ladder_recoverability_L{L}.png   (one per L)
#   - z2_ladder_money_lenB1_multiL.png    (combined scatter, colored by
    L)
#
# Notes:
# - DOFs: qubits on links; link indexing matches Paper D (idx_h/idx_v).
# - Hamiltonian: -g sum X_l - (1/g) sum_p prod Z_l  + Lambda sum_v (1 -
    G_v), with G_v = prod X_l on star(v).
#   The constant +Lambda per star is accounted for as an energy shift
   E_shift.
# - Recoverability uses a *contiguous A-B-C segment in MPS site order*
   (proxy), not BFS collar on the link graph.

import csv
import numpy as np
import scipy.linalg as la
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt

from tenpy.models.model import CouplingMPOModel
from tenpy.networks.site import SpinHalfSite
from tenpy.networks.mps import MPS
from tenpy.algorithms import dmrg


# -------------------------
# Lattice indexing (Paper D convention)
# -------------------------
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 = []
    # horizontals
    for y in range(Ny + 1):
        for x in range(Nx):
            link_endpoints.append((vid(x, y), vid(x + 1, y)))
    # verticals
    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
    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)

    # stars
    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)

    return Nlinks, plaquettes, stars


# -------------------------
# Wilson loops / Creutz proxy
# -------------------------
def rectangle_loop_links(Nx, Ny, x0, y0, R, T):
    links = set()
    for x in range(x0, x0 + R):
        links.add(idx_h(x, y0, Nx, Ny))
        links.add(idx_h(x, y0 + T, Nx, Ny))
    for y in range(y0, y0 + T):
        links.add(idx_v(x0, y, Nx, Ny))
        links.add(idx_v(x0 + R, y, Nx, Ny))
    return sorted(links)

def creutz_ratio(W_RT, W_Rm1Tm1, W_Rm1T, W_RTm1):
    eps = 1e-16
    return -np.log((W_RT * W_Rm1Tm1 + eps) / (W_Rm1T * W_RTm1 + eps))


# -------------------------
# Petz recoverability (dense on ABC)
```

```python
# -------------------------
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

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 fidelity_squared(rho, sigma):
    sr = sqrtm_psd(rho)
    inner = sr @ sigma @ sr
    return (np.trace(sqrtm_psd(inner)).real) ** 2

def petz_recoverability_from_rhoABC(rho_ABC, dA, dB, dC, delta=1e-12):
    dims = [dA, dB, dC]
    rho_AB = partial_trace_rho(rho_ABC, dims, keep=[0, 1])
    rho_B  = partial_trace_rho(rho_ABC, dims, keep=[1])
    rho_BC = partial_trace_rho(rho_ABC, dims, keep=[1, 2])

    rho_B_delta = rho_B + delta * np.eye(dB, dtype=np.complex128)
    O_B = invsqrtm_pd(rho_B_delta)
    sqrt_rho_BC = sqrtm_psd(rho_BC)

    IA = np.eye(dA, dtype=np.complex128)
    IC = np.eye(dC, dtype=np.complex128)

    sandwich = np.kron(IA, O_B)
    middle = sandwich @ rho_AB @ sandwich
    middle = 0.5 * (middle + middle.conj().T)
```

```
    left = np.kron(IA, sqrt_rho_BC)
    sigma = left @ np.kron(middle, IC) @ left
    sigma = 0.5 * (sigma + sigma.conj().T)

    tr_sigma = float(np.trace(sigma).real)
    sigmaN = sigma / tr_sigma

    F = float(np.clip(fidelity_squared(rho_ABC, sigmaN), 0.0, 1.0))
    E = -np.log(F + 1e-16)
    return E, F, tr_sigma


# -------------------------
# TeNPy model: link-qubits on a Chain
# -------------------------
class Z2GaugeLinkChain(CouplingMPOModel):
    default_lattice = "Chain"
    force_default_lattice = True

    def init_sites(self, model_params):
        return SpinHalfSite(conserve=None)

    def init_terms(self, model_params):
        g = float(model_params["g"])
        Lambda = float(model_params["Lambda"])
        plaquettes = model_params["plaquettes"]
        stars = model_params["stars"]

        # Chain with 1-site unit cell: u=0 replicates onsite term on
            all sites
        self.add_onsite(-g, 0, "Sigmax")

        # plaquettes: -(1/g) prod Z
        Jp = -1.0 / g
        for pl in plaquettes:
            term = [("Sigmaz", [int(ell), 0]) for ell in pl]
            self.add_local_term(Jp, term)

        # stars: -Lambda * G_v  (constant +Lambda handled separately as
            energy shift)
        for st in stars:
            if len(st) == 0:
                continue
            term = [("Sigmax", [int(ell), 0]) for ell in st]
            self.add_local_term(-Lambda, term)


def run_ladder(L, g_list, Lambda, delta, lenA, lenC, w_list,
               chi_max=96, max_sweeps=12, svd_min=1e-10, mixer=True):

    Nx = int(L)
    Ny = 2

```

```
    Nlinks, plaquettes, stars = build_lattice(Nx, Ny)
    Nstars = sum(1 for st in stars if len(st) > 0)
    E_shift = Lambda * Nstars

    rows = []
    rec_curves = {}  # g -> {lenB: Erec}

    for g in g_list:
        model_params = {
            "L": Nlinks,
            "bc_MPS": "finite",
            "g": float(g),
            "Lambda": float(Lambda),
            "plaquettes": plaquettes,
            "stars": stars,
        }
        model = Z2GaugeLinkChain(model_params)

        sites = model.lat.mps_sites()
        psi = MPS.from_product_state(
            sites,
            ["up"] * len(sites),
            bc="finite",
            unit_cell_width=len(sites),
        )

        dmrg_params = {
            "mixer": mixer,
            "max_sweeps": int(max_sweeps),
            "trunc_params": {"chi_max": int(chi_max), "svd_min": float(
               svd_min)},
            "max_E_err": 1e-10,
        }
        info = dmrg.run(psi, model, dmrg_params)
        E0 = float(info["E"])
        E0_full = E0 + E_shift

        # Gauss diagnostics
        gv_vals = []
        for st in stars:
            if len(st) == 0:
                continue
            term = [("Sigmax", int(ell)) for ell in st]
            gv_vals.append(float(psi.expectation_value_term(term)))
        mnG = float(np.min(gv_vals))
        avG = float(np.mean(gv_vals))

        print(f"L={L} Nx={Nx} Ny={Ny} g={g}: E0(full)={E0_full:.12f}
           min<Gv>={mnG:.12f}  mean<Gv>={avG:.12f}")

        # Creutz chi(2,2) averaged over x0 where it fits
        sigma_eff = np.nan
        inv_sigma = np.nan
        if Nx >= 2 and Ny >= 2:
```

```python
            chis = []
            for x0 in range(Nx - 1):
                y0 = 0
                W22_links = rectangle_loop_links(Nx, Ny, x0, y0, 2, 2)
                W21_links = rectangle_loop_links(Nx, Ny, x0, y0, 2, 1)
                W12_links = rectangle_loop_links(Nx, Ny, x0, y0, 1, 2)
                W11_links = rectangle_loop_links(Nx, Ny, x0, y0, 1, 1)

                W22 = float(psi.expectation_value_term([("Sigmaz", int(ell)) for ell in W22_links]))
                W21 = float(psi.expectation_value_term([("Sigmaz", int(ell)) for ell in W21_links]))
                W12 = float(psi.expectation_value_term([("Sigmaz", int(ell)) for ell in W12_links]))
                W11 = float(psi.expectation_value_term([("Sigmaz", int(ell)) for ell in W11_links]))

                chis.append(creutz_ratio(W22, W11, W21, W12))
            sigma_eff = float(np.mean(chis))
            inv_sigma = 1.0 / sigma_eff if sigma_eff != 0 else np.nan

        # Recoverability on contiguous A-B-C segment (proxy)
        Erec_by_lenB = {}
        for w in w_list:
            lenB = int(w)
            k = int(lenA + lenB + lenC)

            if k <= 0 or k > 14:
                rows.append([L, g, E0_full, mnG, avG, lenA, lenB, lenC,
                             np.nan, np.nan, np.nan, sigma_eff, inv_sigma, chi_max])
                continue

            start = (Nlinks - k) // 2
            seg = list(range(start, start + k))

            rho_seg = psi.get_rho_segment(seg).to_ndarray().astype(np.complex128)
            rho_ABC = rho_seg.reshape(2**k, 2**k)

            dA = 2**int(lenA)
            dB = 2**int(lenB)
            dC = 2**int(lenC)

            Erec, F, tr_sigma = petz_recoverability_from_rhoABC(rho_ABC, dA, dB, dC, delta=delta)
            Erec_by_lenB[lenB] = Erec

            rows.append([L, g, E0_full, mnG, avG, lenA, lenB, lenC,
                         Erec, F, tr_sigma, sigma_eff, inv_sigma, chi_max])

        rec_curves[g] = Erec_by_lenB

```

```python
    return rows, rec_curves


def plot_recoverability(rec_curves, out_png, title):
    plt.figure(figsize=(7.6, 5.2))
    for g, Erec_by_lenB in rec_curves.items():
        ws = sorted(Erec_by_lenB.keys())
        Es = [max(float(Erec_by_lenB[w]), 1e-16) for w in ws]
        plt.plot(ws, Es, "o--", label=f"g={g}")
    plt.yscale("log")
    plt.xlabel("contiguous buffer size |B| (number of links in MPS
        order)")
    plt.ylabel("E_rec_Petz = -log F")
    plt.title(title)
    plt.grid(True, which="both", ls="--", alpha=0.3)
    plt.legend()
    plt.tight_layout()
    plt.savefig(out_png, dpi=200)
    plt.close()


def plot_money_multiL(csv_path, out_png, fixed_lenB=1):
    # L -> lists
    data = {}
    with open(csv_path, "r", encoding="utf-8") as f:
        rdr = csv.DictReader(f)
        for r in rdr:
            if int(r["lenB"]) != int(fixed_lenB):
                continue
            L = int(r["L"])
            invsig = float(r["inv_sigma_eff"])
            E = r["E_recP"]
            if str(E).lower() == "nan":
                continue
            E = float(E)
            if not np.isfinite(invsig):
                continue
            if L not in data:
                data[L] = ([], [])
            data[L][0].append(invsig)
            data[L][1].append(max(E, 1e-16))

    if not data:
        return

    plt.figure(figsize=(8.0, 5.6))
    markers = ["o", "s", "^", "D", "v", "<", ">"]
    colors = plt.cm.viridis(np.linspace(0.2, 0.85, len(data)))

    for i, (L, (xs, ys)) in enumerate(sorted(data.items())):
        plt.scatter(xs, ys,
                    s=85,
                    marker=markers[i % len(markers)],
                    c=[colors[i]],
```

```
                         edgecolors="black",
                         linewidths=0.5,
                         label=f"L={L}")

    plt.yscale("log")
    plt.xlabel("1 / sigma_eff")
    plt.ylabel(f"E_rec_Petz (contiguous buffer |B|={fixed_lenB})")
    plt.title("Z2 ladder: recoverability vs confinement proxy (multi-L)")
    plt.grid(True, which="both", ls="--", alpha=0.3)
    plt.legend(title="System size")
    plt.tight_layout()
    plt.savefig(out_png, dpi=200)
    plt.close()


def main():
    # ====== Multi-L + moderate g ======
    L_list = [4, 6, 8]
    g_list = [0.5, 0.8, 1.0, 1.2, 1.5, 2.0]

    Lambda = 50.0
    delta = 1e-12

    # contiguous A-B-C sizes (keep k<=14)
    lenA = 4
    lenC = 4
    w_list = [0, 1, 2, 3]

    # DMRG controls
    chi_max = 96
    max_sweeps = 12
    svd_min = 1e-10
    mixer = True

    rows_all = []
    rec_by_L = {}

    for L in L_list:
        print("\\n" + "="*70)
        print(f"RUN: L={L} (Nx={L}, Ny=2), chi_max={chi_max}, sweeps={max_sweeps}")
        print("="*70)
        rows, rec_curves = run_ladder(
            L=L, g_list=g_list, Lambda=Lambda, delta=delta,
            lenA=lenA, lenC=lenC, w_list=w_list,
            chi_max=chi_max, max_sweeps=max_sweeps, svd_min=svd_min, mixer=mixer
        )
        rows_all += rows
        rec_by_L[L] = rec_curves

        plot_recoverability(
            rec_curves,
```

```
            out_png=f"z2_ladder_recoverability_L{L}.png",
            title=f"Z2 ladder Nx={L}, Ny=2: Petz recoverability vs contiguous |B| (DMRG)",
        )

    header = ["L","g","E0_full","min_Gv","mean_Gv","lenA","lenB","lenC",
              "E_recP","F","tr_sigma","sigma_eff","inv_sigma_eff","chi_max"]
    with open("paper_e_results.csv", "w", newline="", encoding="utf-8") as f:
        wr = csv.writer(f)
        wr.writerow(header)
        wr.writerows(rows_all)

    plot_money_multiL("paper_e_results.csv", "z2_ladder_money_lenB1_multiL.png", fixed_lenB=1)

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

A.3 Tensor-network ladders validation script (validate_paper_e.py)

```
# validate_paper_e.py
# Multi-L validator for Paper E v1.1 artifacts.

import os
import sys
import csv

EXPECTED_FILES = [
    "paper_e_reproduce.py",
    "paper_e_results.csv",
    "z2_ladder_money_lenB1_multiL.png",
    "validate_paper_e.py",
]

# Must match paper_e_reproduce.py main()
EXPECTED_L_LIST = [4, 6, 8]
EXPECTED_G_LIST = [0.5, 0.8, 1.0, 1.2, 1.5, 2.0]
EXPECTED_W_LIST = [0, 1, 2, 3]

def die(msg, code=1):
    print("VALIDATION FAILED:", msg)
    sys.exit(code)

def main():
    missing = [f for f in EXPECTED_FILES if not os.path.exists(f)]
    if missing:
        die(f"Missing files: {missing}")

    for L in EXPECTED_L_LIST:
        fn = f"z2_ladder_recoverability_L{L}.png"
```

```
        if not os.path.exists(fn):
            die(f"Missing recoverability plot: {fn}")

    with open("paper_e_results.csv", "r", encoding="utf-8") as f:
        rdr = csv.DictReader(f)
        rows = list(rdr)

    if not rows:
        die("paper_e_results.csv has no rows.")

    required_cols = {
        "L","g","E0_full","min_Gv","mean_Gv","lenA","lenB","lenC",
        "E_recP","F","tr_sigma","sigma_eff","inv_sigma_eff","chi_max"
    }
    if not required_cols.issubset(set(rows[0].keys())):
        die(f"CSV missing required columns: {sorted(required_cols - set(rows[0].keys()))}")

    # Gauge sanity
    for r in rows:
        mn = float(r["min_Gv"])
        av = float(r["mean_Gv"])
        if mn < 0.999 or av < 0.999:
            die(f"Gauss check low: min_Gv={mn}, mean_Gv={av}")

    # Presence of (L,g,lenB) for all expected combos (within float tol for g)
    seen = set()
    for r in rows:
        L = int(r["L"])
        lenB = int(r["lenB"])
        g = float(r["g"])
        for gg in EXPECTED_G_LIST:
            if abs(g - gg) < 1e-12:
                seen.add((L, gg, lenB))

    missing_triplets = [(L, g, w) for L in EXPECTED_L_LIST for g in EXPECTED_G_LIST for w in EXPECTED_W_LIST
                        if (L, g, w) not in seen]
    if missing_triplets:
        die(f"Missing (L,g,lenB) rows for: {missing_triplets[:10]} ... (total missing {len(missing_triplets)})")

    print("VALIDATION OK: files exist; CSV non-empty; columns ok; Gauss ok; expected (L,g,lenB) present.")

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

A.4 Tensor-network ladders convergence check (warm-start) (paper_e_convergence_check_warm

```
#!/usr/bin/env python3
import numpy as np
```

```python
import csv
import importlib.util

def load_module(path, name="pe"):
    spec = importlib.util.spec_from_file_location(name, path)
    mod = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(mod)
    return mod

def compute_metrics(pe, psi, stars, Nx, Ny, lenA, lenC, lenB_list, delta):
    # Gauss check
    gv_vals = []
    for st in stars:
        if len(st) == 0:
            continue
        term = [("Sigmax", int(ell)) for ell in st]
        gv_vals.append(float(psi.expectation_value_term(term)))
    mnG = float(np.min(gv_vals))
    avG = float(np.mean(gv_vals))

    # sigma_eff averaged over x0
    sigma_eff = np.nan
    inv_sigma = np.nan
    if Nx >= 2 and Ny >= 2:
        chis = []
        for x0 in range(Nx - 1):
            y0 = 0
            W22_links = pe.rectangle_loop_links(Nx, Ny, x0, y0, 2, 2)
            W21_links = pe.rectangle_loop_links(Nx, Ny, x0, y0, 2, 1)
            W12_links = pe.rectangle_loop_links(Nx, Ny, x0, y0, 1, 2)
            W11_links = pe.rectangle_loop_links(Nx, Ny, x0, y0, 1, 1)
            W22 = float(psi.expectation_value_term([("Sigmaz", int(ell)) for ell in W22_links]))
            W21 = float(psi.expectation_value_term([("Sigmaz", int(ell)) for ell in W21_links]))
            W12 = float(psi.expectation_value_term([("Sigmaz", int(ell)) for ell in W12_links]))
            W11 = float(psi.expectation_value_term([("Sigmaz", int(ell)) for ell in W11_links]))
            chis.append(pe.creutz_ratio(W22, W11, W21, W12))
        sigma_eff = float(np.mean(chis))
        inv_sigma = 1.0 / sigma_eff if sigma_eff != 0 else np.nan

    # Petz on central segment
    Nlinks = len(psi.sites)
    Erec = {}
    for lenB in lenB_list:
        k = int(lenA + lenB + lenC)
        start = (Nlinks - k) // 2
        seg = list(range(start, start + k))
        rho_seg = psi.get_rho_segment(seg).to_ndarray().astype(np.complex128)
        rho_ABC = rho_seg.reshape(2**k, 2**k)
```

```
        dA, dB, dC = 2**lenA, 2**lenB, 2**lenC
        E, F, tr_sigma = pe.petz_recoverability_from_rhoABC(rho_ABC, dA
            , dB, dC, delta=delta)
        Erec[lenB] = (float(E), float(F), float(tr_sigma), k)

    return mnG, avG, sigma_eff, inv_sigma, Erec

def main():
    pe = load_module("paper_e_reproduce.py", name="pe")

    L = 8
    g = 1.0
    Nx = L
    Ny = 2

    Lambda = 50.0
    delta = 1e-12
    lenA = 4
    lenC = 4
    lenB_list = [1, 2]

    max_sweeps = 20
    svd_min = 1e-10

    # Build lattice
    Nlinks, plaquettes, stars = pe.build_lattice(Nx, Ny)
    Nstars = sum(1 for st in stars if len(st) > 0)
    E_shift = Lambda * Nstars

    model_params = {
        "L": Nlinks,
        "bc_MPS": "finite",
        "g": float(g),
        "Lambda": float(Lambda),
        "plaquettes": plaquettes,
        "stars": stars,
    }
    model = pe.Z2GaugeLinkChain(model_params)

    sites = model.lat.mps_sites()
    psi = pe.MPS.from_product_state(
        sites, ["up"] * len(sites), bc="finite", unit_cell_width=len(
            sites)
    )

    outs = []

    # Stage 1: chi=96
    dmrg_params_96 = {
        "mixer": True,
        "max_sweeps": int(max_sweeps),
        "trunc_params": {"chi_max": 96, "svd_min": float(svd_min)},
        "max_E_err": 1e-10,
    }
```

```python
    info96 = pe.dmrg.run(psi, model, dmrg_params_96)
    E0_full_96 = float(info96["E"]) + E_shift
    mnG, avG, sig, invsig, Erec = compute_metrics(pe, psi, stars, Nx, Ny, lenA, lenC, lenB_list, delta)
    outs.append((96, E0_full_96, mnG, avG, sig, invsig, Erec))

    # Stage 2: continue from SAME psi, increase chi to 192
    dmrg_params_192 = {
        "mixer": False,  # often helps stabilize once close
        "max_sweeps": int(max_sweeps),
        "trunc_params": {"chi_max": 192, "svd_min": float(svd_min)},
        "max_E_err": 1e-10,
    }
    info192 = pe.dmrg.run(psi, model, dmrg_params_192)
    E0_full_192 = float(info192["E"]) + E_shift
    mnG, avG, sig, invsig, Erec = compute_metrics(pe, psi, stars, Nx, Ny, lenA, lenC, lenB_list, delta)
    outs.append((192, E0_full_192, mnG, avG, sig, invsig, Erec))

    # Print
    for chi, E0, mnG, avG, sig, invsig, Erec in outs:
        print("="*70)
        print(f"warm-start result: chi_max={chi}")
        print(f"E0_full={E0:.12f}  min<Gv>={mnG:.12f}  mean<Gv>={avG:.12f}")
        print(f"sigma_eff={sig:.6f}  inv_sigma_eff={invsig:.6f}")
        for lenB, (E, F, tr, k) in Erec.items():
            print(f"  lenB={lenB} (k={k}): E_recP={E:.6e}  F={F:.6e}  tr_sigma={tr:.6f}")

    # CSV
    with open("paper_e_convergence_check_warmstart.csv", "w", newline="", encoding="utf-8") as f:
        wr = csv.writer(f)
        wr.writerow(["L","g","chi_max","E0_full","min_Gv","mean_Gv","sigma_eff","inv_sigma_eff","lenB","k","E_recP","F","tr_sigma"])
        for chi, E0, mnG, avG, sig, invsig, Erec in outs:
            for lenB, (E, F, tr, k) in Erec.items():
                wr.writerow([L, g, chi, E0, mnG, avG, sig, invsig, lenB, k, E, F, tr])

    print("Wrote: paper_e_convergence_check_warmstart.csv")

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