Program A: Semi-Infinite Conditional Mutual Information in the 1D TFIM (iMPS)

Lluis Eriksson
Independent Researcher
lluiseriksson@gmail.com

January 2026

Abstract

We study an information-theoretic notion of locality—approximate quantum Markov behavior—via the conditional mutual information (CMI) $I(A:C \mid B(w))$ in a semi-infinite geometry of the 1D transverse-field Ising model (TFIM). Using infinite matrix product states (iMPS), we compute $I(A:C \mid B(w))$ as a function of the collar width w separating two semi-infinite regions. In a representative gapped point ($h = 1.5$), we observe clean exponential decay and a rapid plateau of the local effective-length estimator, yielding an early-decay length $\xi_{\rm rec}^{(\rm early)}$ comparable to the iMPS transfer-matrix correlation length $\xi_{\rm corr}$. Near criticality ($h = 1.005$), the local estimator increases throughout the accessible range, indicating a pre-asymptotic regime; we therefore report a fixed-window effective length and a window-sensitivity range as a systematic uncertainty. All generated assets used here (two JSONL data streams, the figure, and the LaTeX table snippet) are included in the Overleaf project.

1 Introduction and motivation

A quantitative notion of locality in many-body quantum states is *approximate quantum Markov structure*, commonly diagnosed via the conditional mutual information (CMI) $I(A : C \mid B)$. Small CMI implies the existence of a recovery channel acting on B that approximately reconstructs correlations between A and C (recoverability viewpoint), making $I(A : C \mid B)$ a sharp tool to test how a buffer region B shields distant regions. This perspective has been developed in quantum information theory and connects naturally to physical expectations such as exponential clustering in gapped phases and area-law entanglement [1–5].

Goal (Program A). We perform a controlled iMPS computation of the *semi-infinite* CMI $I(A : C \mid B(w))$ in the 1D transverse-field Ising model (TFIM), where A and C are semi-infinite and B is a contiguous collar of width w. We compare an early-decay length extracted from $I(A : C \mid B(w))$ to the iMPS transfer-matrix correlation length in a gapped regime, and we document pre-asymptotic behavior near criticality when $w_{\max} \ll \xi_{\rm corr}$.

2 Model and numerical setup

We study the TFIM Hamiltonian

$$H = -J \sum_i \sigma_i^z \sigma_{i+1}^z - h \sum_i \sigma_i^x, \tag{1}$$

with $J = 1$. Ground states are computed in the thermodynamic limit using TeNPy iMPS with a two-site unit cell. We report the effective bond dimension $\chi_{\rm eff}$ after canonicalization.

Convention note. All results reported here follow TeNPy's TFIM implementation with $J = 1$ and transverse field parameter $g = h$ in the code; the unit of length is one lattice site, and the unit cell contains two sites. We define $\xi_{\rm corr}$ as the iMPS transfer-matrix correlation length returned by TeNPy's `correlation_length2()` (in lattice-site units).

3 Observable: semi-infinite CMI

We consider a contiguous tripartition A–B–C where A and C are semi-infinite and B is a collar of width w. We evaluate

$$I(A : C \mid B(w)) = S(AB) + S(BC) - S(B) - S(ABC). \tag{2}$$

Operationally, in the semi-infinite geometry considered here (A and C semi-infinite, B finite of width w) and for a globally pure state, the CMI can be evaluated via the finite identity

$$I(A : C \mid B(w)) = 2S_{\rm cut} - S(B(w)), \tag{3}$$

where $S_{\rm cut}$ is the entanglement entropy across a single cut separating a semi-infinite half-chain from its complement, and $S(B(w))$ is the von Neumann entropy of the finite block B. With a two-site unit cell, we average over the two inequivalent cut positions (as done in the Appendix scripts). Equation (3) follows from global purity and the entropy identities $S(AB) = S(C)$, $S(BC) = S(A)$, and $S(ABC) = 0$, yielding a finite cancellation in the semi-infinite geometry with finite $B(w)$.

We also define a local effective-length estimator

$$\xi_{\rm local}(w) := \left[\log I(A : C \mid B(w))(w) - \log I(A : C \mid B(w))(w + 1)\right]^{-1}. \tag{4}$$

A plateau in $\xi_{\rm local}(w)$ indicates an approximately exponential tail.

4 Results

Figure 1 shows $I(A : C \mid B(w))(w)$ and $\xi_{\rm local}(w)$ for the gapped and near-critical runs included in this Overleaf project. Dashed lines show exponential fits on the fixed window $w \in [6, 11]$.

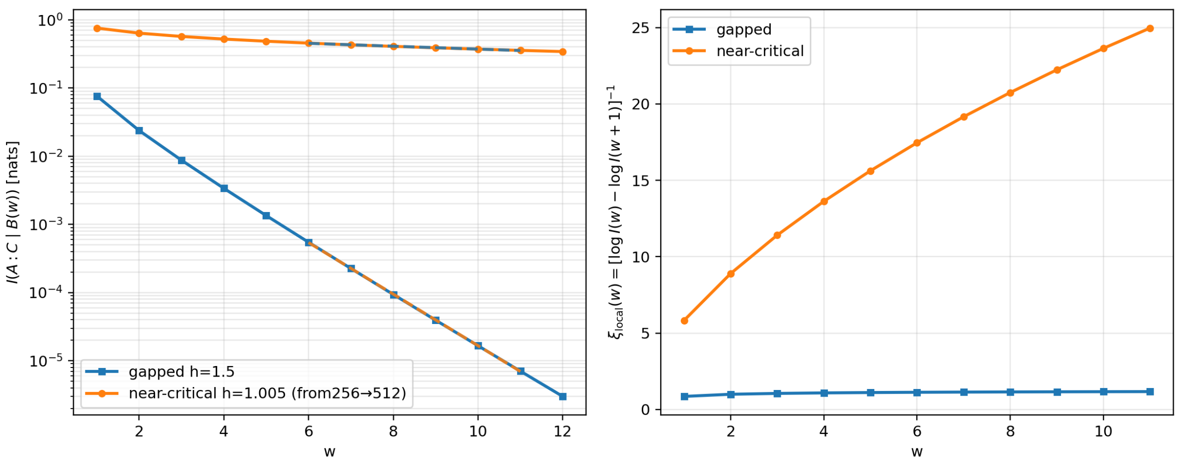

Figure 1: **CMI decay and local effective length.** Left: $I(A : C \mid B(w))(w)$ on a semi-log scale. Right: $\xi_{\rm local}(w)$.

Table 1: **Program A summary (semi-infinite geometry).** $\xi_{\rm rec}^{(\rm early)}$ is extracted from an exponential fit to $I(A:C \mid B(w))$ on a fixed window (main choice: (6, 11)). Window sensitivity is reported across [(6, 11), (6, 12), (7, 12)]. $\chi_{\rm eff}$ is the effective bond dimension after canonicalization. $\xi_{\rm corr}$ is the iMPS transfer-matrix correlation length.

Regime	h	$\chi_{\rm eff}$	$\xi_{\rm corr}$	$\xi_{\rm rec}^{(\rm early)}$
Gapped	1.500	13.000	1.120	1.149
Near-critical (from256→512)	1.005	110.000	87.774	20.485

Window sensitivity. Gapped: $\xi_{\rm rec}^{(\rm early)} \in [1.149, 1.158]$. Near-critical: $\xi_{\rm rec}^{(\rm early)} \in [20.485, 22.017]$.

5 Discussion

Why $\xi_{\rm rec}^{(\rm early)} \approx \xi_{\rm corr}$ in the gapped regime is meaningful. In a gapped 1D phase, connected correlations decay exponentially with distance, and the iMPS transfer matrix encodes this via a finite correlation length $\xi_{\rm corr}$. The observation that $I(A:C \mid B(w))$ decays approximately exponentially and yields an early-decay scale comparable to $\xi_{\rm corr}$ supports the picture that the same transfer-matrix gap governing two-point clustering also controls the onset of approximate Markov structure under a finite buffer.

Why the near-critical regime does not reach the asymptotic tail. Near criticality the correlation length grows rapidly, and our accessible collar widths satisfy $w_{\rm max} \ll \xi_{\rm corr}$. In this pre-asymptotic window, the local estimator $\xi_{\rm local}(w)$ increases with w rather than plateauing, indicating that an effective length extracted from a short window should be interpreted as a scale characterizing the accessible range, not the true asymptotic decay length.

Systematic effects. We report window sensitivity of $\xi_{\rm rec}^{(\rm early)}$ as a minimal systematic uncertainty. A more complete error budget would additionally include finite-χ effects (iMPS truncation), which can be probed by repeating the pipeline at multiple $\chi_{\rm max}$ and examining convergence of $I(A:C \mid B(w))$ at fixed w as $\chi_{\rm max} \to \infty$.

6 Conclusions and outlook

We computed semi-infinite conditional mutual information $I(A:C \mid B(w))$ in the TFIM using iMPS and observed: (i) clean exponential decay and a rapid $\xi_{\rm local}(w)$ plateau in a gapped point, with $\xi_{\rm rec}^{(\rm early)}$ comparable to $\xi_{\rm corr}$; and (ii) a strongly running $\xi_{\rm local}(w)$ near criticality consistent with $w_{\rm max} \ll \xi_{\rm corr}$. Immediate extensions are (a) convergence studies in χ and (b) broader model coverage to test universality of the relation between recoverability length scales and transfer-matrix correlation lengths.

7 Reproducibility

This Overleaf project is populated by uploading four generated files: the figure `moneyplot_CMI_xilocal.png`, the table snippet `summary.tex`, and two JSONL streams containing the points for $I(A:C \mid B(w))$. To regenerate these assets, run the Colab build script and upload the resulting zip (or the four files) to the root of the Overleaf project.

A Appendix: Data sources used in this project

Gapped JSONL: cmi$_h1p5_chi128_semiinf_STREAM.jsonl.$
$Near - criticalJSONL:$

B Appendix: Full scripts (Colab) to regenerate the four uploaded files

This appendix contains the full Colab scripts used to regenerate the four files uploaded to the root of the Overleaf project:

- moneyplot_CMI_xilocal.png
- summary.tex
- cmi_h1p5_chi128_semiinf_STREAM.jsonl
- cmi_from256_h1p005_chi512_semiinf_STREAM.jsonl

The scripts below assume Google Drive is mounted at /content/drive and that PROJECT_DIR is set to your working folder.

B.1 Script 00: Environment, pinned install, and freeze

```
# 00_env_and_install.py
import os, sys, json, platform, subprocess, importlib.util, datetime

def sh(cmd):
    print("+", cmd)
    subprocess.check_call(cmd, shell=True)

PROJECT_DIR = "/content/drive/MyDrive/Ising_Project"
os.makedirs(PROJECT_DIR, exist_ok=True)

# Pin TeNPy to a known tag
if importlib.util.find_spec("tenpy") is None:
    sh("pip -q install git+https://github.com/tenpy/tenpy.git@v1.1.0")

# Ensure core deps exist (Colab already has these; keep minimal to
    avoid conflicts)
for pkg in ["numpy", "matplotlib"]:
    if importlib.util.find_spec(pkg) is None:
        sh(f"pip -q install {pkg}")

import numpy as np
import matplotlib
import tenpy

manifest = {
    "created_utc": datetime.datetime.utcnow().isoformat() + "Z",
    "python": sys.version,
    "platform": platform.platform(),
    "tenpy_version": getattr(tenpy, "__version__", "unknown"),
    "matplotlib_version": matplotlib.__version__,
    "numpy_version": np.__version__,
}
```

```
# Save manifest header (more fields get appended later)
with open(os.path.join(PROJECT_DIR, "repro_manifest.json"), "w") as f:
    json.dump(manifest, f, indent=2, sort_keys=True)

# Freeze full environment (best-effort; large but useful later)
sh(f"pip freeze > {os.path.join(PROJECT_DIR, 'pip_freeze.txt')}")
print("Wrote:", os.path.join(PROJECT_DIR, "repro_manifest.json"))
print("Wrote:", os.path.join(PROJECT_DIR, "pip_freeze.txt"))
```

B.2 Script 01: Ground states (from zero) with a consistent near-critical branch

```
# 01_dmrg_ground_states.py
import os, time
import numpy as np

from tenpy.tools import hdf5_io
from tenpy.models.tf_ising import TFIChain
from tenpy.networks.mps import MPS
from tenpy.algorithms import dmrg

PROJECT_DIR = "/content/drive/MyDrive/Ising_Project"
CKPT_DIR = os.path.join(PROJECT_DIR, "checkpoints")
os.makedirs(CKPT_DIR, exist_ok=True)

H_GAPPED = 1.5
H_NEAR   = 1.005

CHI_GAPPED = 128
CHI_BASE   = 256
CHI_TARGET = 512

CKPT_GAPPED  = os.path.join(CKPT_DIR, "GS_iDMRG_h1p5_chi128.h5")
CKPT_NEAR256 = os.path.join(CKPT_DIR, "GS_iDMRG_h1p005_chi256.h5")
CKPT_NEAR512 = os.path.join(CKPT_DIR, "GS_from256_h1p005_chi512.h5")

def build_model(h):
    return TFIChain({
        "L": 2,
        "J": 1.0,
        "g": float(h),
        "bc_MPS": "infinite",
        "conserve": "best",
    })

def init_psi(M):
    # deterministic product state
    return MPS.from_product_state(M.lat.mps_sites(), ["up"]*M.lat.
       N_sites, bc=M.lat.bc_MPS)

def run_dmrg(M, psi, dmrg_params):
    eng = dmrg.TwoSiteDMRGEngine(psi, M, dmrg_params)
    E0, psi = eng.run()
    psi.canonical_form()
    return float(E0), psi

def save_ckpt(path, psi, meta):
    hdf5_io.save({"psi": psi, **meta}, path)

def load_ckpt(path):
    obj = hdf5_io.load(path)
    psi = obj["psi"] if isinstance(obj, dict) and "psi" in obj else obj
    psi.canonical_form()
    return psi, obj if isinstance(obj, dict) else {}

def chi_eff(psi):
    chi = getattr(psi, "chi", None)
    return int(np.max(np.asarray(chi))) if chi is not None else None

def xi_corr(psi):
    xi = psi.correlation_length2()
    return float(np.max(np.asarray(xi, dtype=float)))

def make_gapped():
    if os.path.exists(CKPT_GAPPED):
        return
    M = build_model(H_GAPPED)
    psi = init_psi(M)
    params = {
        "mixer": True,
        "mixer_params": {"amplitude": 1e-4, "decay": 2.0, "
           disable_after": 40},
        "trunc_params": {"chi_max": int(CHI_GAPPED), "svd_min": 1e-12},
        "max_sweeps": 80,
        "max_E_err": 1e-10,
    }
    t0=time.time()
    E0, psi = run_dmrg(M, psi, params)
    meta = {
        "h": float(H_GAPPED),
        "chi_max": int(CHI_GAPPED),
        "E0": float(E0),
        "chi_eff": chi_eff(psi),
        "xi_corr": xi_corr(psi),
        "runtime_sec": float(time.time()-t0),
    }
    save_ckpt(CKPT_GAPPED, psi, meta)

def make_near_base_256():
    if os.path.exists(CKPT_NEAR256):
        return
    M = build_model(H_NEAR)
    psi = init_psi(M)
    params = {
        "mixer": True,
        "mixer_params": {"amplitude": 1e-4, "decay": 2.0, "
           disable_after": 60},
        "trunc_params": {"chi_max": int(CHI_BASE), "svd_min": 1e-12},
        "max_sweeps": 140,
        "max_E_err": 1e-11,
    }
    t0=time.time()
    E0, psi = run_dmrg(M, psi, params)
    meta = {
        "h": float(H_NEAR),
        "chi_max": int(CHI_BASE),
        "E0": float(E0),
        "chi_eff": chi_eff(psi),
        "xi_corr": xi_corr(psi),
        "runtime_sec": float(time.time()-t0),
    }
    save_ckpt(CKPT_NEAR256, psi, meta)

def ramp_near_to_512_from_256():
    if os.path.exists(CKPT_NEAR512):
        return
    psi256, meta256 = load_ckpt(CKPT_NEAR256)
    M = build_model(H_NEAR)
    params = {
        "mixer": True,
        "mixer_params": {"amplitude": 1e-3, "decay": 1.5, "
           disable_after": 80},
        "trunc_params": {"chi_max": int(CHI_TARGET), "svd_min": 1e-12},
        "chi_list": {0: 128, 20: 256, 40: 384, 60: 512},
        "max_sweeps": 180,
        "max_E_err": 1e-12,
    }
    t0=time.time()
    E0, psi = run_dmrg(M, psi256, params)
    meta = {
        "h": float(H_NEAR),
        "chi_max": int(CHI_TARGET),
        "E0": float(E0),
        "chi_eff": chi_eff(psi),
        "xi_corr": xi_corr(psi),
        "source_ckpt": CKPT_NEAR256,
        "source_meta": {k: meta256.get(k) for k in ["E0","chi_eff","
           xi_corr","chi_max","h"]},
        "runtime_sec": float(time.time()-t0),
    }
    save_ckpt(CKPT_NEAR512, psi, meta)

make_gapped()
make_near_base_256()
ramp_near_to_512_from_256()

print("Ready checkpoints:")
print(" -", CKPT_GAPPED)
print(" -", CKPT_NEAR256)
print(" -", CKPT_NEAR512)
```

B.3 Script 02: Semi-infinite CMI to JSONL

```
# 02_cmi_jsonl.py
import os, json, time
import numpy as np
from tenpy.tools import hdf5_io

PROJECT_DIR = "/content/drive/MyDrive/Ising_Project"
DATA_DIR = os.path.join(PROJECT_DIR, "data")
CKPT_DIR = os.path.join(PROJECT_DIR, "checkpoints")
os.makedirs(DATA_DIR, exist_ok=True)
```

```
W_MAX = 12

CKPT_GAPPED  = os.path.join(CKPT_DIR, "GS_iDMRG_h1p5_chi128.h5")
CKPT_NEAR512 = os.path.join(CKPT_DIR, "GS_from256_h1p005_chi512.h5")

OUT_GAPPED = os.path.join(DATA_DIR, "cmi_h1p5_chi128_semiinf_STREAM.
    jsonl")
OUT_NEAR   = os.path.join(DATA_DIR, "
    cmi_from256_h1p005_chi512_semiinf_STREAM.jsonl")

def load_psi(path):
    obj = hdf5_io.load(path)
    psi = obj["psi"] if isinstance(obj, dict) and "psi" in obj else obj
    psi.canonical_form()
    return psi

def S_cut_mean(psi):
    return float(np.mean(np.asarray(psi.entanglement_entropy(), dtype=
        float)))

def S_segment(psi, w, first_site):
    seg = list(range(w))
    out = psi.entanglement_entropy_segment(segment=seg, first_site=[int
        (first_site)])
    return float(out[0])

def CMI_semiinf(psi, w):
    # operational implementation of I(A:C|B)=2*S_cut - S(B),
    # averaged over the two inequivalent cuts for a 2-site unit cell.
    Sc = S_cut_mean(psi)
    I0 = 2.0*Sc - S_segment(psi, w, 0)
    I1 = 2.0*Sc - S_segment(psi, w, (1 % max(1, psi.L)))
    return float(0.5*(I0+I1)), float(I0), float(I1), float(Sc)

def write_stream(out_path, ckpt_path, psi):
    if os.path.exists(out_path):
        os.remove(out_path)
    for w in range(1, W_MAX+1):
        t0=time.time()
        I, I0, I1, Sc = CMI_semiinf(psi, w)
        rec = {
            "type": "point",
            "mode": "semi_infinite",
            "ckpt": ckpt_path,
            "w": int(w),
            "I": float(I),
            "I0": float(I0),
            "I1": float(I1),
            "S_cut_mean": float(Sc),
            "dt_sec": float(time.time()-t0),
        }
        with open(out_path, "a") as f:
            f.write(json.dumps(rec) + "\n")
    print("Wrote:", out_path)

psi_g = load_psi(CKPT_GAPPED)
psi_n = load_psi(CKPT_NEAR512)
```

```
write_stream(OUT_GAPPED, CKPT_GAPPED, psi_g)
write_stream(OUT_NEAR,   CKPT_NEAR512, psi_n)
```

B.4 Script 03: Figure and LaTeX summary table snippet

```
# 03_figure_and_summary.py
import os, json, math
import numpy as np
import matplotlib.pyplot as plt
from tenpy.tools import hdf5_io

PROJECT_DIR = "/content/drive/MyDrive/Ising_Project"
DATA_DIR   = os.path.join(PROJECT_DIR, "data")
FIG_PATH   = os.path.join(PROJECT_DIR, "moneyplot_CMI_xilocal.png")  #
   ROOT for Overleaf
TEX_PATH   = os.path.join(PROJECT_DIR, "summary.tex")             #
   ROOT for Overleaf
CKPT_DIR   = os.path.join(PROJECT_DIR, "checkpoints")

JSONL_G = os.path.join(DATA_DIR, "cmi_h1p5_chi128_semiinf_STREAM.jsonl"
   )
JSONL_N = os.path.join(DATA_DIR, "
   cmi_from256_h1p005_chi512_semiinf_STREAM.jsonl")

CKPT_G   = os.path.join(CKPT_DIR, "GS_iDMRG_h1p5_chi128.h5")
CKPT_N   = os.path.join(CKPT_DIR, "GS_from256_h1p005_chi512.h5")

FLOOR = 1e-14
FIT_WINDOWS = [(6,11), (6,12), (7,12)]  # main choice is first

def read_points(path):
    w_to_I = {}
    with open(path, "r") as f:
        for line in f:
            if not line.strip():
                continue
            o = json.loads(line)
            if o.get("type") != "point":
                continue
            if "w" in o and "I" in o:
                w_to_I[int(o["w"])] = float(o["I"])
    ws = np.array(sorted(w_to_I.keys()), dtype=int)
    Is = np.array([w_to_I[w] for w in ws], dtype=float)
    return ws, Is

def xi_local(ws, Is):
    Is = np.maximum(np.asarray(Is, float), FLOOR)
    ws = np.asarray(ws, int)
    wloc, xiloc = [], []
    for i in range(len(ws)-1):
        if ws[i+1] != ws[i] + 1:
            continue
        d = np.log(Is[i]) - np.log(Is[i+1])
        if d > 0:
            wloc.append(ws[i])
            xiloc.append(1.0/d)
```

```python
    return np.array(wloc), np.array(xiloc)

def exp_fit(ws, Is, wmin, wmax):
    ws = np.asarray(ws, float)
    Is = np.asarray(Is, float)
    m = (ws>=wmin) & (ws<=wmax) & (Is>10*FLOOR)
    ww = ws[m]
    yy = np.log(Is[m])
    if len(ww) < 3:
        return None
    A = np.vstack([np.ones_like(ww), -ww]).T
    logA, inv_xi = np.linalg.lstsq(A, yy, rcond=None)[0]
    if inv_xi <= 0:
        return None
    return {"wmin": int(wmin), "wmax": int(wmax), "xi": float(1.0/
        inv_xi), "logA": float(logA)}

def fit_curve(fit, wgrid):
    wgrid = np.asarray(wgrid, float)
    return np.exp(fit["logA"] - wgrid/fit["xi"])

def xi_range(fits):
    xs = [f["xi"] for f in fits if f is not None]
    return (min(xs), max(xs)) if xs else (None, None)

def load_psi(path):
    obj = hdf5_io.load(path)
    psi = obj["psi"] if isinstance(obj, dict) and "psi" in obj else obj
    psi.canonical_form()
    return psi

def chi_eff(psi):
    chi = getattr(psi, "chi", None)
    return int(np.max(np.asarray(chi))) if chi is not None else None

def xi_corr(psi):
    xi = psi.correlation_length2()
    return float(np.max(np.asarray(xi, dtype=float)))

def fmt(x, nd=3):
    if x is None: return "nan"
    if isinstance(x, float) and (math.isnan(x) or math.isinf(x)):
        return "nan"
    return f"{x:.{nd}f}"

# Load data
wg, Ig = read_points(JSONL_G)
wn, In = read_points(JSONL_N)

fits_g = [exp_fit(wg, Ig, a, b) for (a,b) in FIT_WINDOWS]
fits_n = [exp_fit(wn, In, a, b) for (a,b) in FIT_WINDOWS]
if fits_g[0] is None or fits_n[0] is None:
    raise RuntimeError("Main fit window failed; check JSONL coverage/
        values.")

wlg, xlg = xi_local(wg, Ig)
wln, xln = xi_local(wn, In)
```

```
# Figure (ROOT)
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(11.2,4.5))

ax1.semilogy(wg, np.maximum(Ig, FLOOR), "s-", ms=4, lw=2.0, label="
    gapped h=1.5")
ax1.semilogy(wn, np.maximum(In, FLOOR), "o-", ms=4, lw=2.0, label="near
    -critical h=1.005 (from256512)")

for fit, color in [(fits_g[0], "C1"), (fits_n[0], "C0")]:
    wgrid = np.arange(fit["wmin"], fit["wmax"]+1)
    ax1.semilogy(wgrid, np.maximum(fit_curve(fit, wgrid), FLOOR), "--",
        lw=2.0, color=color, alpha=0.85)

ax1.set_xlabel("w")
ax1.set_ylabel(r"$I(A:C\mid B(w))$ [nats]")
ax1.grid(True, which="both", alpha=0.3)
ax1.legend()

ax2.plot(wlg, xlg, "s-", ms=4, lw=2.0, label="gapped")
ax2.plot(wln, xln, "o-", ms=4, lw=2.0, label="near-critical")
ax2.set_xlabel("w")
ax2.set_ylabel(r"$\xi_{\rm local}(w)=[\log I(w)-\log I(w+1)]^{-1}$")
ax2.grid(True, alpha=0.3)
ax2.legend()

plt.tight_layout()
plt.savefig(FIG_PATH, dpi=220)
plt.close(fig)
print("Wrote:", FIG_PATH)

# Load checkpoints for chi_eff and xi_corr
psi_g = load_psi(CKPT_G)
psi_n = load_psi(CKPT_N)

chi_g = chi_eff(psi_g)
chi_n = chi_eff(psi_n)

xi_g = xi_corr(psi_g)
xi_n = xi_corr(psi_n)

g_lo, g_hi = xi_range(fits_g)
n_lo, n_hi = xi_range(fits_n)

TABLE_TEX = rf"""
\begin{{table}}[H]
\centering
\caption{{\textbf{{Program A summary (semi-infinite geometry).}}
$\xi_{{\mathrm{{rec}}}}^{{(\mathrm{{early}})}}$ is extracted from an
    exponential fit to $I(A:C\mid B(w))$ on a fixed window (main choice:
     {FIT_WINDOWS[0]}).
Window sensitivity is reported across {FIT_WINDOWS}.
$\chi_{{\mathrm{{eff}}}}$ is the effective bond dimension after
    canonicalization.
$\xi_{{\mathrm{{corr}}}}$ is the iMPS transfer-matrix correlation
    length.}}
\label{{tab:programA}}
\sisetup{{round-mode=places,round-precision=3}}
```

```
\begin{{tabular}}{{l S[table-format=1.3] S[table-format=3.0] S[table-
    format=3.3] S[table-format=2.3]}}
\toprule
Regime & {{$h$}} & {{$\chi_{{\mathrm{{eff}}}}$}} & {{$\xi_{{\mathrm{{
    corr}}}}$}} & {{$\xi_{{\mathrm{{rec}}}}^{{(\mathrm{{early}})}}$}} \\
\midrule
Gapped & 1.500 & {chi_g} & {fmt(xi_g,3)} & {fmt(fits_g[0]["xi"],3)} \\
Near-critical (from256$\to$512) & 1.005 & {chi_n} & {fmt(xi_n,3)} & {
    fmt(fits_n[0]["xi"],3)} \\
\bottomrule
\end{{tabular}}

\vspace{{0.4em}}
\begin{{minipage}}{{0.94\linewidth}}
\small
\textbf{{Window sensitivity.}}
Gapped: $\xi_{{\mathrm{{rec}}}}^{{(\mathrm{{early}})}}\in[{fmt(g_lo,3)
    },{fmt(g_hi,3)}]$.
Near-critical: $\xi_{{\mathrm{{rec}}}}^{{(\mathrm{{early}})}}\in[{fmt(
    n_lo,3)},{fmt(n_hi,3)}]$.
\end{{minipage}}
\end{{table}}
""".lstrip()

with open(TEX_PATH, "w") as f:
    f.write(TABLE_TEX)
print("Wrote:", TEX_PATH)
```

B.5 Script 04: Package the four Overleaf-root files and write hashes

```
# 04_package_and_manifest.py
import os, json, hashlib, zipfile, datetime

PROJECT_DIR = "/content/drive/MyDrive/Ising_Project"

FILES = [
    ("moneyplot_CMI_xilocal.png", os.path.join(PROJECT_DIR, "
        moneyplot_CMI_xilocal.png")),
    ("summary.tex",               os.path.join(PROJECT_DIR, "summary.
        tex")),
    ("cmi_h1p5_chi128_semiinf_STREAM.jsonl",
                                 os.path.join(PROJECT_DIR, "data/
                                    cmi_h1p5_chi128_semiinf_STREAM.
                                    jsonl")),
    ("cmi_from256_h1p005_chi512_semiinf_STREAM.jsonl",
                                 os.path.join(PROJECT_DIR, "data/
                                    cmi_from256_h1p005_chi512_semiinf_STREAM
                                    .jsonl")),
]

ZIP_PATH = os.path.join(PROJECT_DIR, "overleaf_assets_exact.zip")
MANIFEST_PATH = os.path.join(PROJECT_DIR, "repro_manifest.json")

def sha256(path):
    h = hashlib.sha256()
    with open(path, "rb") as f:
        for chunk in iter(lambda: f.read(1024*1024), b""):
            h.update(chunk)
    return h.hexdigest()

# verify files
out = []
for arc, src in FILES:
    if not os.path.exists(src):
        raise FileNotFoundError(src)
    size = os.path.getsize(src)
    if size == 0:
        raise RuntimeError(f"0-byte file: {src}")
    out.append({"arcname": arc, "src": src, "bytes": size, "sha256":
       sha256(src)})

# zip root-clean (exactly 4 files at root)
if os.path.exists(ZIP_PATH):
    os.remove(ZIP_PATH)

with zipfile.ZipFile(ZIP_PATH, "w", compression=zipfile.ZIP_DEFLATED)
   as z:
    for item in out:
        z.write(item["src"], arcname=item["arcname"])

print("Wrote zip:", ZIP_PATH, "bytes:", os.path.getsize(ZIP_PATH))

# append manifest info
manifest = {}
if os.path.exists(MANIFEST_PATH):
    with open(MANIFEST_PATH, "r") as f:
        manifest = json.load(f)

manifest["outputs_utc"] = datetime.datetime.utcnow().isoformat() + "Z"
manifest["overleaf_root_files"] = out
manifest["zip"] = {"path": ZIP_PATH, "bytes": os.path.getsize(ZIP_PATH)
    , "sha256": sha256(ZIP_PATH)}

with open(MANIFEST_PATH, "w") as f:
    json.dump(manifest, f, indent=2, sort_keys=True)

print("Updated manifest:", MANIFEST_PATH)
print("ZIP sha256:", manifest["zip"]["sha256"])
```

References

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

[2] F. G. S. L. Brandão, A. W. Harrow, J. Oppenheim, and S. Strelchuk, Quantum conditional mutual information, reconstructed states, and state redistribution, *Physical Review Letters* 115, 050501 (2015); arXiv:1411.4921.

[3] D. Sutter, O. Fawzi, and R. Renner, Universal recovery map for approximate Markov chains, arXiv:1504.07251 (2015).

[4] D. Sutter, M. Berta, and M. Tomamichel, Multivariate trace inequalities, *Communications in Mathematical Physics* 352, 37--58 (2017).

[5] F. G. S. L. Brandão and M. Horodecki, Exponential decay of correlations implies area law, *Communications in Mathematical Physics* 333, 761--798 (2015); arXiv:1206.2947.