# Python Code

```
"""
Unified Observer-Inference Analysis (Final)
============================================
One observer. One predictive model. Three questions about its performance.

M: Does history help me predict beyond the present alone?
   → Positive = "this system remembers"

L: Is my prediction improving over time?
   → Positive = "this system is learning"

D: Does my prediction suddenly fail?
   → Positive = "this system made a decision"

All three derived from the same rolling prediction machinery.
All four substrates in one run.
Multi-observer hierarchy test included.

Author: D. Neale / Goleudy.ai
Date: March 2026
For: "Memory Without Storage" and "The Philosophy Loop" papers
"""

import numpy as np
from collections import Counter
import csv
import itertools
import sys

# ============================================================
# ENGINES
# ============================================================

def ca_step(state, rule):
    n = len(state)
    new = np.zeros(n, dtype=int)
    bits = [(rule >> i) & 1 for i in range(8)]
    for i in range(n):
        nb = (state[(i-1)%n] << 2) | (state[i] << 1) | state[(i+1)%n]
        new[i] = bits[nb]
    return new

def run_ca(rule, width=101, steps=300, seed=42):
    rng = np.random.RandomState(seed)
    state = rng.randint(0, 2, width)
    grid = np.zeros((steps, width), dtype=int)
    grid[0] = state
    for t in range(1, steps):
        grid[t] = ca_step(grid[t-1], rule)
    return grid

def gol_step(g):
    r, c = g.shape
    new = np.zeros_like(g)
    for i in range(r):
        for j in range(c):
            n = sum(g[(i+di)%r,(j+dj)%c] for di in [-1,0,1] for dj in [-1,0,1]) - g[i,j]
            new[i,j] = 1 if (g[i,j]==1 and n in (2,3)) or (g[i,j]==0 and n==3) else 0
    return new

```

```python
def run_gol(pattern, gs=50, steps=400):
    grid = np.zeros((gs,gs), dtype=int)
    p = np.array(pattern, dtype=int)
    pr, pc = p.shape
    sr, sc = gs//2-pr//2, gs//2-pc//2
    grid[sr:sr+pr, sc:sc+pc] = p
    hist = np.zeros((steps,gs,gs), dtype=int)
    hist[0] = grid
    for t in range(1, steps):
        hist[t] = gol_step(hist[t-1])
    return hist

def laplacian(f):
    return np.roll(f,1,0)+np.roll(f,-1,0)+np.roll(f,1,1)+np.roll(f,-1,1)-4*f

def run_gs(F, k, gs=64, steps=5000, sample=10, seed=42):
    rng = np.random.RandomState(seed)
    U = np.ones((gs,gs)); V = np.zeros((gs,gs))
    sz = gs//10; r = gs//2
    U[r-sz:r+sz,r-sz:r+sz] = 0.5+0.02*rng.randn(2*sz,2*sz)
    V[r-sz:r+sz,r-sz:r+sz] = 0.25+0.02*rng.randn(2*sz,2*sz)
    U = np.clip(U,0,1); V = np.clip(V,0,1)
    n_s = steps//sample
    Uh = np.zeros((n_s,gs,gs)); Vh = np.zeros((n_s,gs,gs))
    idx = 0
    for s in range(steps):
        if s % sample == 0 and idx < n_s:
            Uh[idx]=U; Vh[idx]=V; idx+=1
        Lu=laplacian(U); Lv=laplacian(V); uvv=U*V*V
        U = np.clip(U+0.16*Lu-uvv+F*(1-U),0,1)
        V = np.clip(V+0.08*Lv+uvv-(F+k)*V,0,1)
    return Uh[:idx], Vh[:idx]

def bubble_sort_history(arr):
    a = arr.copy(); n = len(a)
    history = [a.copy()]
    for i in range(n):
        sw = False
        for j in range(n-i-1):
            if a[j]>a[j+1]: a[j],a[j+1]=a[j+1],a[j]; sw=True
        history.append(a.copy())
        if not sw: break
    return history

def self_sort_history(arr, max_passes=None):
    a = arr.copy(); n = len(a)
    if max_passes is None: max_passes = n*2
    history = [a.copy()]
    rng = np.random.RandomState(42)
    for _ in range(max_passes):
        indices = rng.permutation(n-1); sw = False
        for j in indices:
            if a[j]>a[j+1]: a[j],a[j+1]=a[j+1],a[j]; sw=True
        history.append(a.copy())
        if not sw: break
    return history

def self_sort_defects_history(arr, defects, max_passes=None):
    a = arr.copy(); n = len(a)
    if max_passes is None: max_passes = n*3
    history = [a.copy()]
    rng = np.random.RandomState(42); ds = set(defects)
    for _ in range(max_passes):
        indices = rng.permutation(n-1); sw = False
```

```
        for j in indices:
            if j in ds or (j+1) in ds: continue
            if a[j]>a[j+1]: a[j],a[j+1]=a[j+1],a[j]; sw=True
        history.append(a.copy())
        if not sw: break
    return history

def selection_sort_history(arr):
    a = arr.copy(); n = len(a)
    history = [a.copy()]
    for i in range(n):
        mi = i
        for j in range(i+1,n):
            if a[j]<a[mi]: mi=j
        a[i],a[mi]=a[mi],a[i]
        history.append(a.copy())
    return history

# ============================================================
# EMISSIONS
# ============================================================

def emit_ca(grid):
    T, W = grid.shape
    E = np.zeros((T, 4))
    for t in range(T):
        row = grid[t]
        E[t,0] = np.mean(row)
        blocks = [(row[i]<<2)|(row[i+1]<<1)|row[i+2] for i in range(W-2)]
        counts = Counter(blocks); total = len(blocks)
        E[t,1] = -sum((c/total)*np.log2(c/total) for c in counts.values() if c>0)
        E[t,2] = np.sum(np.abs(np.diff(row)))/(W-1)
        E[t,3] = np.mean(grid[t]!=grid[t-1]) if t>0 else 0
    return E

def emit_gol(hist):
    T, R, C = hist.shape
    E = np.zeros((T, 4))
    for t in range(T):
        g = hist[t]
        E[t,0] = np.mean(g)
        blocks = []
        for r in range(R-1):
            for c in range(C-1):
                blocks.append((g[r,c]<<3)|(g[r,c+1]<<2)|(g[r+1,c]<<1)|g[r+1,c+1])
        counts = Counter(blocks); tb = len(blocks)
        E[t,1] = -sum((c/tb)*np.log2(c/tb) for c in counts.values() if c>0)
        live = np.sum(g)
        if live > 0:
            bd = 0
            for r in range(R):
                for c in range(C):
                    if g[r,c]==1:
                        for dr in [-1,0,1]:
                            for dc in [-1,0,1]:
                                if dr==0 and dc==0: continue
                                if g[(r+dr)%R,(c+dc)%C]==0: bd+=1; break
                            else: continue
                            break
            E[t,2] = bd/(R*C)
        E[t,3] = np.mean(hist[t]!=hist[t-1]) if t>0 else 0
    return E

def emit_gs(Uh, Vh):
```

```
    T = len(Uh)
    E = np.zeros((T, 6))
    for t in range(T):
        U, V = Uh[t], Vh[t]
        E[t,0] = np.mean(U)
        E[t,1] = np.mean(V)
        E[t,2] = np.var(V)
        dVx = V[1:,:]-V[:-1,:]; dVy = V[:,1:]-V[:,:-1]
        E[t,3] = np.mean(dVx**2)+np.mean(dVy**2)
        E[t,4] = np.mean(np.abs(Vh[t]-Vh[t-1])) if t>0 else 0
        Vd = np.clip((V*8).astype(int),0,7)
        counts = np.bincount(Vd.flatten(), minlength=8)
        probs = counts/counts.sum()
        E[t,5] = -sum(p*np.log2(p) for p in probs if p>0)
    return E

def emit_sort(history):
    T = len(history); n = len(history[0])
    E = np.zeros((T, 5))
    sorted_arr = np.sort(history[0])
    for t in range(T):
        a = history[t]
        E[t,0] = sum(1 for i in range(n-1) if a[i]<=a[i+1])/(n-1)
        sp = {v:i for i,v in enumerate(sorted_arr)}
        E[t,1] = np.mean([abs(i-sp.get(a[i],i)) for i in range(n)])/n
        inv = sum(1 for i in range(n) for j in range(i+1,n) if a[i]>a[j])
        E[t,2] = inv/(n*(n-1)/2) if n>1 else 0
        E[t,3] = np.mean(history[t]!=history[t-1]) if t>0 else 0
        bs = max(1,n//10)
        bm = [np.mean(a[i:i+bs]) for i in range(0,n,bs)]
        if len(bm)>1:
            bm = np.array(bm); bm_n = bm/(bm.sum()+1e-10)
            E[t,4] = -sum(p*np.log2(p) for p in bm_n if p>0)
    return E

# ============================================================
# UNIFIED PREDICTION-BASED OBSERVER
# ============================================================

def observer_predict(E, window=10):
    """
    The observer's predictive model.

    Two strategies computed simultaneously:
      Present-only: predict E(t) from E(t-1) alone
      History: predict E(t) from E(t-window:t) mean

    Returns:
      surprise: prediction error at each step (history-based)
      surprise_present: prediction error using present only
      Both as arrays with NaN where not computable.
    """
    T, d = E.shape
    w = min(window, max(2, T // 10))

    surprise = np.full(T, np.nan)
    surprise_present = np.full(T, np.nan)

    for t in range(w, T):
        # History-based prediction: rolling mean
        pred_hist = np.mean(E[t-w:t], axis=0)
        surprise[t] = np.sqrt(np.mean((E[t] - pred_hist)**2))

        # Present-only prediction: just the previous step
```

```
        pred_present = E[t-1]
        surprise_present[t] = np.sqrt(np.mean((E[t] - pred_present)**2))

    return surprise, surprise_present


def measure_MLD(E, window=10):
    """
    Unified M/L/D from one predictive model.

    M: history helps beyond present alone
    L: surprise decreasing over time (gated by activity)
    D: surprise spikes relative to local baseline

    Returns dict with M, L, D_rate, D_mag, D_times, and diagnostics.
    """
    T, d = E.shape
    w = min(window, max(2, T // 10))

    surprise, surprise_present = observer_predict(E, window=w)

    valid = np.where(~np.isnan(surprise) & ~np.isnan(surprise_present))[0]

    if len(valid) < 10:
        return {'M': 0, 'L': 0, 'D_rate': 0, 'D_mag': 0, 'D_times': [],
                'D_centre': 0.5, 'surprise_early': 0, 'surprise_late': 0,
                'memory_early': 0, 'memory_late': 0, 'alive': False}

    # ----- M: Does history help? -----
    # M = mean(surprise_present - surprise_history) where positive
    # If history-based prediction is better, the difference is positive
    memory_signal = surprise_present[valid] - surprise[valid]
    M = max(0.0, np.mean(memory_signal))

    # Also compute M for early vs late to check scaling
    third = max(3, len(valid) // 3)
    early_v = valid[:third]
    late_v = valid[-third:]

    memory_early = max(0.0, np.mean(surprise_present[early_v] - surprise[early_v]))
    memory_late = max(0.0, np.mean(surprise_present[late_v] - surprise[late_v]))

    # ----- L: Is surprise decreasing? -----
    s_early = np.mean(surprise[early_v])
    s_late = np.mean(surprise[late_v])

    if s_early > 1e-10:
        reduction = (s_early - s_late) / s_early
    else:
        reduction = 0.0

    # Activity gate: system must still be active in late phase
    late_em = E[late_v]
    early_em = E[early_v]
    late_act = np.mean(np.abs(np.diff(late_em, axis=0))) if len(late_em) > 1 else 0
    early_act = np.mean(np.abs(np.diff(early_em, axis=0))) if len(early_em) > 1 else 0
    floor = 0.01 * early_act if early_act > 1e-10 else 1e-10
    alive = late_act > floor

    L = max(0.0, reduction) if alive else 0.0

    # ----- D: Surprise spikes -----
    threshold = 2.0
    d_window = min(15, len(valid) // 4)
```

```
    surprises_list = []
    d_times = []

    s_vals = surprise[valid]
    for i in range(d_window, len(s_vals)):
        recent = s_vals[i-d_window:i]
        med = np.median(recent)
        mad = np.median(np.abs(recent - med))
        if mad > 1e-10:
            z = (s_vals[i] - med) / mad
        elif med > 1e-10:
            z = s_vals[i] / med if s_vals[i] > 1e-10 else 0
        else:
            z = 0
        if z > threshold:
            surprises_list.append(z)
            d_times.append(valid[i])

    n_eligible = len(s_vals) - d_window
    D_rate = len(surprises_list) / n_eligible if n_eligible > 0 else 0
    D_mag = np.median(surprises_list) if surprises_list else 0
    D_centre = np.mean(d_times) / T if d_times else 0.5

    return {
        'M': M, 'L': L, 'D_rate': D_rate, 'D_mag': D_mag,
        'D_times': d_times, 'D_centre': D_centre,
        'surprise_early': s_early, 'surprise_late': s_late,
        'memory_early': memory_early, 'memory_late': memory_late,
        'alive': alive,
    }


def derive_character(r):
    """Observer's narrative from M/L/D pattern."""
    M, L, D_rate, D_centre = r['M'], r['L'], r['D_rate'], r['D_centre']
    if D_rate > 0 and L > 0.05 and D_centre < 0.4:
        return 'purposive'
    elif D_rate > 0.15 and L < 0.01:
        return 'chaotic'
    elif M < 0.001 and L < 0.001 and D_rate < 0.001:
        return 'static'
    elif D_rate > 0 or L > 0:
        return 'mixed'
    else:
        return 'inert'

# ============================================================
# CLASSIFICATIONS
# ============================================================

WOLFRAM_CLASS = {
    1: [0,8,32,40,64,96,128,136,160,168,192,224,234,235,238,239,
        248,249,250,251,252,253,254,255],
    2: [1,2,3,4,5,6,7,9,10,11,12,13,14,15,19,23,24,25,26,27,
        28,29,33,34,35,36,37,38,42,43,44,46,50,51,56,57,58,62,
        65,66,67,68,69,72,73,74,76,77,78,94,104,108,130,132,134,
        138,140,142,152,154,156,162,164,170,172,176,178,184,200,204,232],
    3: [18,22,30,45,60,75,86,89,90,101,102,105,106,109,120,121,
        122,126,129,146,149,150,151,153,161,165,169,182,195,225],
    4: [41,54,110,124,131,133,137,193]
}

def get_wc(rule):
    for c, rules in WOLFRAM_CLASS.items():
```

```
        if rule in rules: return c
    return 2

GOL_PATTERNS = {
    'Block': ([[1,1],[1,1]], 'still_life'),
    'Beehive': ([[0,1,1,0],[1,0,0,1],[0,1,1,0]], 'still_life'),
    'Loaf': ([[0,1,1,0],[1,0,0,1],[0,1,0,1],[0,0,1,0]], 'still_life'),
    'Blinker': ([[1,1,1]], 'oscillator'),
    'Toad': ([[0,1,1,1],[1,1,1,0]], 'oscillator'),
    'Beacon': ([[1,1,0,0],[1,1,0,0],[0,0,1,1],[0,0,1,1]], 'oscillator'),
    'Pulsar': ([[0,0,1,1,1,0,0,0,1,1,1,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0],
                [1,0,0,0,0,1,0,1,0,0,0,0,1],[1,0,0,0,0,1,0,1,0,0,0,0,1],
                [1,0,0,0,0,1,0,1,0,0,0,0,1],[0,0,1,1,1,0,0,0,1,1,1,0,0],
                [0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,1,1,1,0,0,0,1,1,1,0,0],
                [1,0,0,0,0,1,0,1,0,0,0,0,1],[1,0,0,0,0,1,0,1,0,0,0,0,1],
                [1,0,0,0,0,1,0,1,0,0,0,0,1],[0,0,0,0,0,0,0,0,0,0,0,0,0],
                [0,0,1,1,1,0,0,0,1,1,1,0,0]], 'oscillator'),
    'Pentadecathlon': ([[0,0,1,0,0,0,0,1,0,0],[1,1,0,1,1,1,1,0,1,1],
                        [0,0,1,0,0,0,0,1,0,0]], 'oscillator'),
    'Glider': ([[0,1,0],[0,0,1],[1,1,1]], 'spaceship'),
    'LWSS': ([[0,1,0,0,1],[1,0,0,0,0],[1,0,0,0,1],[1,1,1,1,0]], 'spaceship'),
    'MWSS': ([[0,0,1,0,0,0],[1,0,0,0,1,0],[0,0,0,0,0,1],[1,0,0,0,0,1],
              [0,1,1,1,1,1]], 'spaceship'),
    'HWSS': ([[0,0,1,1,0,0,0],[1,0,0,0,0,1,0],[0,0,0,0,0,0,1],
              [1,0,0,0,0,0,1],[0,1,1,1,1,1,1]], 'spaceship'),
    'R_pentomino': ([[0,1,1],[1,1,0],[0,1,0]], 'methuselah'),
    'Acorn': ([[0,1,0,0,0,0,0],[0,0,0,1,0,0,0],[1,1,0,0,1,1,1]], 'methuselah'),
    'Diehard': ([[0,0,0,0,0,0,1,0],[1,1,0,0,0,0,0,0],[0,1,0,0,0,1,1,1]], 'methuselah'),
}

GS_REGIMES = {
    'GS_death': (0.078, 0.061), 'GS_spots': (0.035, 0.065),
    'GS_stripes': (0.040, 0.065), 'GS_moving': (0.014, 0.054),
    'GS_mitosis': (0.028, 0.062), 'GS_pulsing': (0.025, 0.060),
    'GS_coral': (0.062, 0.061), 'GS_chaos': (0.026, 0.051),
    'GS_worms': (0.054, 0.063), 'GS_sparse': (0.030, 0.062),
}

# ============================================================
# MAIN
# ============================================================

def main():
    print("=" * 70)
    print("  UNIFIED OBSERVER-INFERENCE ANALYSIS")
    print("  One observer. One predictive model. Three questions.")
    print("  Four substrates. Results for paper.")
    print("=" * 70)

    all_results = []

    # ---- 1. CELLULAR AUTOMATA ----
    print("\n  === CELLULAR AUTOMATA (256 rules) ===")
    ca_results = []
    for rule in range(256):
        if (rule+1) % 64 == 0 or rule == 0:
            print(f"    Rule {rule:3d} ({rule+1}/256)")
        grid = run_ca(rule, steps=300)
        E = emit_ca(grid)
        r = measure_MLD(E)
        r['name'] = f'CA_{rule:03d}'
        r['category'] = f'class_{get_wc(rule)}'
        r['wolfram_class'] = get_wc(rule)
        r['character'] = derive_character(r)
```

```
        ca_results.append(r)
        all_results.append(r)

    # ---- 2. GAME OF LIFE ----
    print("\n  === GAME OF LIFE (18 patterns) ===")
    gol_results = []
    for name, (pattern, ptype) in GOL_PATTERNS.items():
        gs = 60 if name in ('Acorn',) else 50
        st = 400 if ptype == 'methuselah' else 300
        print(f"    {name}")
        hist = run_gol(pattern, gs=gs, steps=st)
        E = emit_gol(hist)
        r = measure_MLD(E)
        r['name'] = name
        r['category'] = ptype
        r['character'] = derive_character(r)
        gol_results.append(r)
        all_results.append(r)

    # ---- 3. GRAY-SCOTT ----
    print("\n  === GRAY-SCOTT (10 regimes) ===")
    gs_results = []
    for name, (F, k) in GS_REGIMES.items():
        print(f"    {name} (F={F}, k={k})")
        Uh, Vh = run_gs(F, k, gs=64, steps=5000, sample=10)
        E = emit_gs(Uh, Vh)
        r = measure_MLD(E)
        r['name'] = name
        r['category'] = 'gray_scott'
        r['character'] = derive_character(r)
        gs_results.append(r)
        all_results.append(r)

    # ---- 4. SORTING ----
    print("\n  === SORTING ALGORITHMS (7 variants) ===")
    sort_results = []
    rng = np.random.RandomState(123)
    arr = rng.permutation(30)

    sort_systems = {
        'already_sorted': ([np.arange(30)]*20, 'control'),
        'bubble_sort': (bubble_sort_history(arr.copy()), 'top_down'),
        'self_sort': (self_sort_history(arr.copy()), 'bottom_up'),
        'self_sort_defects': (self_sort_defects_history(arr.copy(), [7,15,22]), 'defective'),
        'selection_sort': (selection_sort_history(arr.copy()), 'top_down'),
        'reverse_bubble': (bubble_sort_history(np.arange(30)[::-1].copy()), 'worst_case'),
    }

    for name, (history, stype) in sort_systems.items():
        history_np = [np.array(h) for h in history]
        if len(history_np) < 5:
            print(f"    {name}: too short ({len(history_np)} steps), skipping")
            continue
        print(f"    {name}")
        E = emit_sort(history_np)
        r = measure_MLD(E, window=5)
        r['name'] = name
        r['category'] = stype
        r['character'] = derive_character(r)
        sort_results.append(r)
        all_results.append(r)

    # ============================================================
    # SAVE RESULTS
```

```
    # ============================================================

    fields = ['name','category','M','L','D_rate','D_mag','D_centre',
              'character','surprise_early','surprise_late',
              'memory_early','memory_late','alive']

    for path, data, extra_fields in [
        ('/home/claude/final_ca.csv', ca_results, ['wolfram_class']),
        ('/home/claude/final_gol.csv', gol_results, []),
        ('/home/claude/final_gs.csv', gs_results, []),
        ('/home/claude/final_sort.csv', sort_results, []),
    ]:
        with open(path, 'w', newline='') as f:
            writer = csv.DictWriter(f, fieldnames=fields+extra_fields)
            writer.writeheader()
            for r in data:
                row = {k: r.get(k,'') for k in fields+extra_fields}
                writer.writerow(row)

    # ============================================================
    # REPORTS
    # ============================================================

    # --- CA by class ---
    print("\n" + "=" * 70)
    print("  CA RESULTS BY WOLFRAM CLASS")
    print("=" * 70)
    for cls in [1, 2, 3, 4]:
        cr = [r for r in ca_results if r['wolfram_class'] == cls]
        if not cr: continue
        M = np.mean([r['M'] for r in cr])
        L = np.mean([r['L'] for r in cr])
        D = np.mean([r['D_rate'] for r in cr])
        Ms = np.std([r['M'] for r in cr])
        Ls = np.std([r['L'] for r in cr])
        Ds = np.std([r['D_rate'] for r in cr])
        chars = Counter(r['character'] for r in cr)
        print(f"\n  Class {cls} ({len(cr)} rules):")
        print(f"    M = {M:.4f} ± {Ms:.4f}   L = {L:.4f} ± {Ls:.4f}   D = {D:.4f} ± {Ds:.4f}")
        print(f"    Characters: {dict(chars)}")

    # --- GoL by type ---
    print("\n" + "=" * 70)
    print("  GoL RESULTS BY TYPE")
    print("=" * 70)
    for ptype in ['still_life','oscillator','spaceship','methuselah']:
        pr = [r for r in gol_results if r['category'] == ptype]
        if not pr: continue
        print(f"\n  {ptype} ({len(pr)}):")
        print(f"    M = {np.mean([r['M'] for r in pr]):.4f}  "
              f"L = {np.mean([r['L'] for r in pr]):.4f}  "
              f"D = {np.mean([r['D_rate'] for r in pr]):.4f}")

    # --- GoL individual ---
    print("\n  Individual GoL patterns:")
    for r in gol_results:
        print(f"    {r['name']:20s} ({r['category']:12s}): "
              f"M={r['M']:.4f} L={r['L']:.4f} D={r['D_rate']:.4f} [{r['character']}]")

    # --- GS ---
    print("\n" + "=" * 70)
    print("  GRAY-SCOTT RESULTS")
    print("=" * 70)
```

```
    for r in sorted(gs_results, key=lambda x: -(x['M']*x['L']*x['D_rate'])**(1/3) if x['M']>0 and x['L']>0 and x['D_rate']>0 else 0):
        ov = (r['M']*r['L']*r['D_rate'])**(1/3) if r['M']>0 and r['L']>0 and r['D_rate']>0 else 0
        print(f"  {r['name']:15s}: M={r['M']:.4f} L={r['L']:.4f} D={r['D_rate']:.4f} ov={ov:.4f} [{r['character']}]")

    # --- Sorting ---
    print("\n" + "=" * 70)
    print("  SORTING RESULTS")
    print("=" * 70)
    for r in sort_results:
        print(f"  {r['name']:20s}: M={r['M']:.4f} L={r['L']:.4f} D={r['D_rate']:.4f} Dc={r['D_centre']:.2f} [{r['character']}]")

    # --- Notable rules ---
    print("\n" + "=" * 70)
    print("  NOTABLE CA RULES")
    print("=" * 70)
    for rule in [0, 4, 30, 54, 110, 184]:
        r = next(x for x in ca_results if x['name'] == f'CA_{rule:03d}')
        print(f"  Rule {rule:3d} (Class {r['wolfram_class']}): "
              f"M={r['M']:.4f} L={r['L']:.4f} D={r['D_rate']:.4f} Dc={r['D_centre']:.2f} [{r['character']}]")

    # --- Cross-substrate MnLnD overlap top 20 ---
    print("\n" + "=" * 70)
    print("  CROSS-SUBSTRATE TOP 20 (MnLnD overlap)")
    print("=" * 70)

    scored = []
    for r in all_results:
        if r['M'] > 0 and r['L'] > 0 and r['D_rate'] > 0:
            # Normalise within each substrate? No — use raw geometric mean
            ov = (r['M'] * r['L'] * r['D_rate']) ** (1/3)
        else:
            ov = 0
        scored.append((ov, r))
    scored.sort(key=lambda x: -x[0])

    for i, (ov, r) in enumerate(scored[:20]):
        print(f"  {i+1:2d}. {r['name']:25s} ({r['category']:14s}): "
              f"ov={ov:.4f} M={r['M']:.4f} L={r['L']:.4f} D={r['D_rate']:.4f} [{r['character']}]")

    # ============================================================
    # MULTI-OBSERVER HIERARCHY TEST
    # ============================================================

    print("\n" + "=" * 70)
    print("  MULTI-OBSERVER COGNITIVE HIERARCHY TEST")
    print("  Same system, different access levels")
    print("=" * 70)

    # Select test systems
    test_systems = [
        ('CA Rule 54', emit_ca(run_ca(54, steps=300))),
        ('CA Rule 30', emit_ca(run_ca(30, steps=300))),
        ('GoL R-pentomino', emit_gol(run_gol([[0,1,1],[1,1,0],[0,1,0]], steps=400))),
    ]

    # Add GS mitosis
    print("  Generating GS Mitosis for hierarchy test...")
    Uh, Vh = run_gs(0.028, 0.062, gs=64, steps=5000, sample=10)
    test_systems.append(('GS Mitosis', emit_gs(Uh, Vh)))

```

```
    # Add bubble sort
    arr = np.random.RandomState(123).permutation(30)
    test_systems.append(('Bubble Sort', emit_sort([np.array(h) for h in bubble_sort_history(arr)])))

    for sys_name, E in test_systems:
        n_feat = E.shape[1]
        print(f"\n  {sys_name} ({n_feat} features):")
        print(f"    {'Observer':12s} {'D_obs':>5s} {'M':>8s} {'L':>8s} {'D_rate':>8s}")

        # Full access
        r_full = measure_MLD(E)
        print(f"    {'Full':12s} {n_feat:5d} {r_full['M']:8.4f} {r_full['L']:8.4f}
{r_full['D_rate']:8.4f}")

        # Pairs (average)
        if n_feat >= 3:
            pair_M, pair_L, pair_D = [], [], []
            for combo in itertools.combinations(range(n_feat), 2):
                r_p = measure_MLD(E[:, list(combo)])
                pair_M.append(r_p['M']); pair_L.append(r_p['L']); pair_D.append(r_p['D_rate'])
            print(f"    {'Pairs (avg)':12s} {2:5d} {np.mean(pair_M):8.4f} {np.mean(pair_L):8.4f}
{np.mean(pair_D):8.4f}")

        # Singles (average)
        sing_M, sing_L, sing_D = [], [], []
        for f in range(n_feat):
            r_s = measure_MLD(E[:, f:f+1])
            sing_M.append(r_s['M']); sing_L.append(r_s['L']); sing_D.append(r_s['D_rate'])
        print(f"    {'Singles (avg)':12s} {1:5d} {np.mean(sing_M):8.4f} {np.mean(sing_L):8.4f}
{np.mean(sing_D):8.4f}")

        # Direction check
        m_up = np.mean(sing_M) > r_full['M'] + 0.001
        l_up = np.mean(sing_L) > r_full['L'] + 0.001
        d_up = np.mean(sing_D) > r_full['D_rate'] + 0.001

        arrows = []
        if m_up: arrows.append("M↑")
        if l_up: arrows.append("L↑")
        if d_up: arrows.append("D↑")

        if arrows:
            print(f"    Less access → more: {', '.join(arrows)}")
        else:
            print(f"    No hierarchy effect detected")

    print("\n" + "=" * 70)
    print("  ANALYSIS COMPLETE")
    print("=" * 70)
    print(f"  Total systems analysed: {len(all_results)}")
    print(f"  Files saved: final_ca.csv, final_gol.csv, final_gs.csv, final_sort.csv")


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