Python Code

```
"""
Multi-Observer Hierarchy: Detailed Investigation
================================================
For R-pentomino and CA Rule 54, L *decreases* with restricted access.
This goes against the prediction. Why?

Hypothesis A: Measurement limitation — our L measure can't detect
learning from a single feature when learning is in the correlations.

Hypothesis B: Genuine result — for these systems, more access really
does reveal more learning, and the hierarchy doesn't hold for L.

Investigation: Break down L by individual feature to see which features
carry the learning signal and which don't.

Author: D. Neale / Goleudy.ai
Date: March 2026
"""

import numpy as np
from collections import Counter

# ---- Engines (compact) ----
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

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

```

```python
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

# ---- 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 = [(g[r,c]<<3)|(g[r,c+1]<<2)|(g[r+1,c]<<1)|g[r+1,c+1]
                  for r in range(R-1) for c in range(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

```

```python
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

# ---- L measure ----
def measure_L_detailed(E, window=10):
    """Returns L and detailed diagnostics including per-step surprise."""
    T, d = E.shape
    w = min(window, max(2, T//10))

    surprise = np.full(T, np.nan)
    for t in range(w, T):
        pred = np.mean(E[t-w:t], axis=0)
        surprise[t] = np.sqrt(np.mean((E[t]-pred)**2))

    valid = np.where(~np.isnan(surprise))[0]
    if len(valid) < 10:
        return 0.0, {'surprise': surprise, 's_early': 0, 's_late': 0,
                     'reduction': 0, 'alive': False}

    third = max(3, len(valid)//3)
    early_v = valid[:third]
    late_v = valid[-third:]

    s_early = np.mean(surprise[early_v])
    s_late = np.mean(surprise[late_v])

    reduction = (s_early - s_late) / s_early if s_early > 1e-10 else 0

    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

    return L, {
        'surprise': surprise, 's_early': s_early, 's_late': s_late,
        'reduction': reduction, 'alive': alive,
        'early_act': early_act, 'late_act': late_act,
    }


# ============================================================
# INVESTIGATION
# ============================================================

def investigate_system(name, E, feature_names):
    """
    For one system, compute L using:
    - All features together
    - Each individual feature alone
    - Each pair of features

    Report which features carry the learning signal.
    """
    n_feat = E.shape[1]

    print(f"\n{'='*70}")
    print(f"  {name}")
    print(f"  {n_feat} features: {', '.join(feature_names)}")
    print(f"{'='*70}")

    # Full access
    L_full, d_full = measure_L_detailed(E)
    print(f"\n  FULL ACCESS (all {n_feat} features):")
    print(f"    L = {L_full:.4f}")
    print(f"    Surprise: early={d_full['s_early']:.6f} → late={d_full['s_late']:.6f}")
    print(f"    Reduction: {d_full['reduction']:.4f}  Alive: {d_full['alive']}")

    # Each feature alone
    print(f"\n  INDIVIDUAL FEATURES:")
    individual_L = {}
    for f in range(n_feat):
        E_single = E[:, f:f+1]
        L_s, d_s = measure_L_detailed(E_single)
        individual_L[feature_names[f]] = L_s
        marker = " ← carries learning" if L_s > 0.1 else (" ← no learning" if L_s < 0.01 else "")
        print(f"    {feature_names[f]:20s}: L={L_s:.4f}  "
              f"surprise {d_s['s_early']:.6f}→{d_s['s_late']:.6f}  "
              f"alive={d_s['alive']}{marker}")

    # How does averaging work?
    avg_individual = np.mean(list(individual_L.values()))
    print(f"\n  Average of individual L values: {avg_individual:.4f}")
    print(f"  Full-access L:                  {L_full:.4f}")

    if L_full > avg_individual + 0.05:
        print(f"  → Full > Average: LEARNING IS IN THE CORRELATIONS")
        print(f"    The observer needs multiple features to see the learning")
        print(f"    because surprise reduction comes from inter-feature structure")
    elif avg_individual > L_full + 0.05:
        print(f"  → Average > Full: DILUTION EFFECT")
        print(f"    Some features carry learning, others add noise")
        print(f"    The observer with fewer (right) features sees MORE learning")
    else:
        print(f"  → Approximately equal: learning is visible in individual features")

```

```
    # Which features help and which hurt?
    print(f"\n  FEATURE CONTRIBUTION ANALYSIS:")
    for fn, lv in sorted(individual_L.items(), key=lambda x: -x[1]):
        if lv > L_full:
            print(f"    {fn:20s}: L={lv:.4f}  (ABOVE full-access L — this feature alone is better)")
        elif lv > 0.01:
            print(f"    {fn:20s}: L={lv:.4f}  (contributes to learning signal)")
        else:
            print(f"    {fn:20s}: L={lv:.4f}  (no learning signal — adds noise)")

    # Pair analysis for the failing systems
    if n_feat >= 3:
        print(f"\n  PAIR ANALYSIS (how L changes as we ADD features):")
        # Find best individual feature
        best_feat = max(individual_L, key=individual_L.get)
        best_idx = feature_names.index(best_feat)

        print(f"    Best single feature: {best_feat} (L={individual_L[best_feat]:.4f})")

        for f in range(n_feat):
            if f == best_idx:
                continue
            pair = sorted([best_idx, f])
            E_pair = E[:, pair]
            L_p, _ = measure_L_detailed(E_pair)
            direction = "↑" if L_p > individual_L[best_feat] + 0.01 else ("↓" if L_p <
individual_L[best_feat] - 0.01 else "=")
            print(f"    + {feature_names[f]:20s}: L={L_p:.4f}  {direction}")

    return L_full, individual_L


def main():
    print("=" * 70)
    print("  Multi-Observer Hierarchy: Detailed Investigation")
    print("  Why does L decrease with less access for some systems?")
    print("=" * 70)

    # The two failing systems
    print("\n\n  ============ SYSTEMS WHERE HIERARCHY FAILS ============")

    # R-pentomino
    hist = run_gol([[0,1,1],[1,1,0],[0,1,0]], gs=50, steps=400)
    E = emit_gol(hist)
    investigate_system("GoL R-pentomino (HIERARCHY FAILS: L drops with less access)",
                       E, ['density', 'spatial_entropy', 'boundary_density', 'activity'])

    # CA Rule 54
    grid = run_ca(54, steps=300)
    E = emit_ca(grid)
    investigate_system("CA Rule 54 (HIERARCHY FAILS: L drops with less access)",
                       E, ['density', 'spatial_entropy', 'boundaries', 'activity'])

    # The three confirming systems
    print("\n\n  ============ SYSTEMS WHERE HIERARCHY HOLDS ============")

    # Bubble sort
    rng = np.random.RandomState(123)
    arr = rng.permutation(30)
    E = emit_sort([np.array(h) for h in bubble_sort_history(arr)])
    investigate_system("Bubble Sort (HIERARCHY HOLDS: L increases with less access)",
                       E, ['sortedness', 'displacement', 'inversions', 'activity', 'block_entropy'])

    # GS Mitosis
```

```
    Uh, Vh = run_gs(0.028, 0.062, gs=64, steps=5000, sample=10)
    E = emit_gs(Uh, Vh)
    investigate_system("GS Mitosis (HIERARCHY HOLDS: L increases with less access)",
                       E, ['mean_U', 'mean_V', 'var_V', 'grad_energy', 'activity', 'entropy'])

    # CA Rule 30
    grid = run_ca(30, steps=300)
    E = emit_ca(grid)
    investigate_system("CA Rule 30 (HIERARCHY HOLDS: L increases with less access)",
                       E, ['density', 'spatial_entropy', 'boundaries', 'activity'])

    # ============================================================
    # SYNTHESIS
    # ============================================================
    print("\n\n" + "=" * 70)
    print("  SYNTHESIS: WHY DOES THE HIERARCHY SOMETIMES FAIL?")
    print("=" * 70)
    print("""
  The investigation should reveal one of two things:

  1. CORRELATION EFFECT: For failing systems, L(full) >> average(L(individual)).
     Learning is in the inter-feature correlations. Removing features removes
     the signal. This is a MEASUREMENT LIMITATION — a more sophisticated
     predictor (e.g., multivariate regression) might capture learning from
     fewer features by exploiting temporal cross-correlations.

  2. DILUTION EFFECT: For confirming systems, some features carry strong L
     while others carry zero L. The full-access observer averages over all,
     diluting the signal. The single-feature observer who happens to watch
     the right feature sees MORE learning.

  If (1) is the explanation, the hierarchy holds in principle but our simple
  rolling-mean predictor can't see it with restricted access. The fix is a
  better predictor.

  If (2) is the explanation, the hierarchy genuinely holds — the average
  single-feature observer sees more learning than the full-access observer
  because dilution outweighs correlation.

  In either case, the hierarchy is about the OBSERVER'S MODEL QUALITY, not
  about the system. Different model choices would produce different hierarchy
  results. This is itself a demonstration of the thesis: what the observer
  infers depends on the observer's modelling choices, not on the system.
    """)


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