Python Code

```
"""
Dual Memory Measure Comparison
==============================
Two different operationalisations of "does this system remember?"

M_pred: Model obsolescence. A predictor trained on early emissions
        performs worse on late emissions than a predictor trained on
        late emissions. The system moved and stayed moved.

M_cohen: Baseline shift. Cohen's d between early and late emission
         distributions. Large effect size = persistent regime change.

If both agree on the ranking, the result is robust.

Tested across selected systems from all four substrates.

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

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 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 = [(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

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

# ============================================================
# TWO M MEASURES
# ============================================================

def measure_M_pred(E, window=10):
    """
    M_pred: Model obsolescence.

    Build a predictor from early emissions (rolling mean).
    Apply it to late emissions.
    Compare error to a predictor built from late emissions.

```

```
    If the early model fails on late data, the system has moved
    persistently — it "remembers" its history.
    """
    T, d = E.shape
    third = T // 3
    if third < 10:
        return 0.0, {}

    early = E[5:5+third]
    late = E[T-third:T]
    w = min(window, third // 3)
    if w < 2:
        return 0.0, {}

    # Early model: rolling mean trained on early data
    # Compute its prediction error on late data
    early_mean = np.mean(early, axis=0)
    early_std = np.std(early, axis=0)

    # "Early model predicting late data": how surprised is the early
    # model by each late emission?
    error_early_on_late = np.mean([np.sqrt(np.mean((late[t] - early_mean)**2))
                                   for t in range(len(late))])

    # "Late model predicting late data": rolling mean within late window
    error_late_on_late = []
    for t in range(w, len(late)):
        pred = np.mean(late[t-w:t], axis=0)
        error_late_on_late.append(np.sqrt(np.mean((late[t] - pred)**2)))
    error_late_on_late = np.mean(error_late_on_late) if error_late_on_late else 0

    # M = how much worse is the early model on late data?
    M = max(0.0, error_early_on_late - error_late_on_late)

    details = {
        'error_early_on_late': error_early_on_late,
        'error_late_on_late': error_late_on_late,
    }
    return M, details


def measure_M_cohen(E):
    """
    M_cohen: Baseline shift (Cohen's d).

    Compare early and late emission distributions.
    Large effect size = persistent regime change.
    The system "remembers" — it's in a different place now.
    """
    T, d = E.shape
    third = T // 3
    if third < 5:
        return 0.0, {}

    early = E[5:5+third]
    late = E[T-third:T]

    # Cohen's d for each feature, then average absolute d
    cohens = []
    for feat in range(d):
        e_early = early[:, feat]
        e_late = late[:, feat]

        mean_diff = abs(np.mean(e_late) - np.mean(e_early))
```

```
253
254         # Pooled standard deviation
255         n1, n2 = len(e_early), len(e_late)
256         var1, var2 = np.var(e_early, ddof=1), np.var(e_late, ddof=1)
257
258         if n1 + n2 < 4:
259             continue
260
261         pooled_std = np.sqrt(((n1-1)*var1 + (n2-1)*var2) / (n1+n2-2))
262
263         if pooled_std > 1e-10:
264             cohens.append(mean_diff / pooled_std)
265         else:
266             # Both distributions are constant
267             if mean_diff > 1e-10:
268                 cohens.append(10.0)  # massive shift from one constant to another
269             else:
270                 cohens.append(0.0)  # same constant
271
272     M = np.mean(cohens) if cohens else 0.0
273
274     details = {
275         'cohens_per_feature': cohens,
276         'mean_cohen': M,
277     }
278     return M, details
279
280
281 # Also measure L and D for context
282 def measure_L(E, window=10):
283     T, d = E.shape
284     w = min(window, max(2, T//10))
285     surprise = np.full(T, np.nan)
286     for t in range(w, T):
287         pred = np.mean(E[t-w:t], axis=0)
288         surprise[t] = np.sqrt(np.mean((E[t]-pred)**2))
289     valid = np.where(~np.isnan(surprise))[0]
290     if len(valid) < 10: return 0.0
291     third = max(3, len(valid)//3)
292     s_early = np.mean(surprise[valid[:third]])
293     s_late = np.mean(surprise[valid[-third:]])
294     if s_early < 1e-10: return 0.0
295     reduction = (s_early - s_late) / s_early
296     late_em = E[valid[-third:]]
297     early_em = E[valid[:third]]
298     late_act = np.mean(np.abs(np.diff(late_em,axis=0))) if len(late_em)>1 else 0
299     early_act = np.mean(np.abs(np.diff(early_em,axis=0))) if len(early_em)>1 else 0
300     floor = 0.01*early_act if early_act>1e-10 else 1e-10
301     return max(0.0, reduction) if late_act > floor else 0.0
302
303 def measure_D(E, window=10, threshold=2.0):
304     T, d = E.shape
305     if T < window+5: return 0.0
306     changes = np.sqrt(np.sum(np.diff(E,axis=0)**2, axis=1))
307     surprises = []
308     for t in range(window, len(changes)):
309         recent = changes[t-window:t]
310         med = np.median(recent); mad = np.median(np.abs(recent-med))
311         if mad>1e-10: z=(changes[t]-med)/mad
312         elif med>1e-10: z=changes[t]/med if changes[t]>1e-10 else 0
313         else: z=0
314         if z>threshold: surprises.append(z)
315     n = len(changes)-window
316     return len(surprises)/n if n>0 else 0.0
```

```python


# ============================================================
# TEST SYSTEMS
# ============================================================

def main():
    print("=" * 75)
    print("  Dual Memory Measure Comparison")
    print("  M_pred (model obsolescence) vs M_cohen (baseline shift)")
    print("  Do they agree on which systems 'remember'?")
    print("=" * 75)

    systems = []

    # CA
    print("\n  Generating CA systems...")
    for rule, desc in [(0,'C1 dies'), (4,'C2 static'), (30,'C3 chaos'),
                       (54,'C4 complex'), (110,'C4 universal'), (184,'C2 traffic')]:
        E = emit_ca(run_ca(rule, steps=300))
        systems.append((f'CA {rule} ({desc})', E))

    # GoL
    print("  Generating GoL systems...")
    patterns = {
        'Block': ([[1,1],[1,1]], 'still'),
        'Blinker': ([[1,1,1]], 'osc'),
        '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]], 'osc'),
        'Glider': ([[0,1,0],[0,0,1],[1,1,1]], 'ship'),
        'R_pentomino': ([[0,1,1],[1,1,0],[0,1,0]], 'meth'),
        '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]], 'meth'),
    }
    for name, (pat, ptype) in patterns.items():
        gs = 60 if name == 'Acorn' else 50
        st = 400 if ptype == 'meth' else 300
        E = emit_gol(run_gol(pat, gs=gs, steps=st))
        systems.append((f'GoL {name} ({ptype})', E))

    # Gray-Scott
    print("  Generating Gray-Scott systems...")
    for name, F, k in [('death',0.078,0.061), ('spots',0.035,0.065),
                       ('stripes',0.040,0.065), ('mitosis',0.028,0.062),
                       ('pulsing',0.025,0.060), ('coral',0.062,0.061),
                       ('chaos',0.026,0.051)]:
        Uh, Vh = run_gs(F, k, gs=64, steps=5000, sample=10)
        E = emit_gs(Uh, Vh)
        systems.append((f'GS {name}', E))

    # Sorting
    print("  Generating sorting systems...")
    rng = np.random.RandomState(123)
    arr = rng.permutation(30)

    E = emit_sort([np.array(h) for h in [np.arange(30)]*20])
    systems.append(('Sort: already sorted', E))

    E = emit_sort([np.array(h) for h in bubble_sort_history(arr.copy())])
    systems.append(('Sort: bubble', E))
```

```
381
382     E = emit_sort([np.array(h) for h in selection_sort_history(arr.copy())])
383     systems.append(('Sort: selection', E))
384
385     E = emit_sort([np.array(h) for h in bubble_sort_history(np.arange(30)[::-1].copy())])
386     systems.append(('Sort: reverse bubble', E))
387
388     # ============================================================
389     # RUN BOTH MEASURES
390     # ============================================================
391
392     results = []
393
394     print(f"\n{'='*75}")
395     print(f"  {'System':30s}  {'M_pred':>7s}  {'M_cohen':>7s}  {'L':>7s}  {'D':>7s}")
396     print(f"  {'-'*70}")
397
398     for name, E in systems:
399         M_p, M_p_details = measure_M_pred(E)
400         M_c, M_c_details = measure_M_cohen(E)
401         L = measure_L(E)
402         D = measure_D(E)
403
404         results.append({
405             'name': name, 'M_pred': M_p, 'M_cohen': M_c,
406             'L': L, 'D': D,
407             'M_p_details': M_p_details, 'M_c_details': M_c_details,
408         })
409
410         print(f"  {name:30s}  {M_p:7.4f}  {M_c:7.4f}  {L:7.4f}  {D:7.4f}")
411
412     # ============================================================
413     # CORRELATION BETWEEN THE TWO M MEASURES
414     # ============================================================
415
416     M_pred_vals = np.array([r['M_pred'] for r in results])
417     M_cohen_vals = np.array([r['M_cohen'] for r in results])
418
419     # Rank correlation (Spearman) — more robust than Pearson for this
420     from scipy.stats import spearmanr
421     try:
422         rho, pval = spearmanr(M_pred_vals, M_cohen_vals)
423         print(f"\n  Spearman correlation between M_pred and M_cohen: ρ = {rho:.4f} (p = {pval:.4f})")
424     except:
425         # Manual rank correlation if scipy not available
426         ranks_p = np.argsort(np.argsort(M_pred_vals))
427         ranks_c = np.argsort(np.argsort(M_cohen_vals))
428         rho = np.corrcoef(ranks_p, ranks_c)[0,1]
429         print(f"\n  Rank correlation between M_pred and M_cohen: r = {rho:.4f}")
430
431     # ============================================================
432     # RANKING COMPARISON
433     # ============================================================
434
435     print(f"\n{'='*75}")
436     print(f"  RANKING COMPARISON")
437     print(f"{'='*75}")
438
439     ranked_pred = sorted(results, key=lambda x: -x['M_pred'])
440     ranked_cohen = sorted(results, key=lambda x: -x['M_cohen'])
441
442     print(f"\n  {'Rank':>4s}  {'By M_pred':30s}  {'M_p':>7s}  {'By M_cohen':30s}  {'M_c':>7s}")
443     print(f"  {'-'*85}")
444
```

```
    for i in range(len(results)):
        rp = ranked_pred[i]
        rc = ranked_cohen[i]
        print(f"  {i+1:4d}  {rp['name']:30s}  {rp['M_pred']:7.4f}  "
              f"{rc['name']:30s}  {rc['M_cohen']:7.4f}")

    # ============================================================
    # M_PRED DETAILS for interesting cases
    # ============================================================

    print(f"\n{'='*75}")
    print(f"  M_PRED DETAILS (model obsolescence)")
    print(f"{'='*75}")

    for r in sorted(results, key=lambda x: -x['M_pred'])[:10]:
        d = r['M_p_details']
        if d:
            print(f"\n  {r['name']}:")
            print(f"    Early model on late data: {d.get('error_early_on_late',0):.6f}")
            print(f"    Late model on late data:  {d.get('error_late_on_late',0):.6f}")
            print(f"    Obsolescence (M_pred):    {r['M_pred']:.6f}")

    # ============================================================
    # CROSS-CHECK: MnLnD with each M measure
    # ============================================================

    print(f"\n{'='*75}")
    print(f"  MnLnD OVERLAP (using each M measure)")
    print(f"{'='*75}")

    print(f"\n  Using M_pred:")
    scored_p = [(((r['M_pred']*r['L']*r['D'])**(1/3) if r['M_pred']>0 and r['L']>0 and r['D']>0 else 0),
r)
                for r in results]
    scored_p.sort(key=lambda x: -x[0])
    for ov, r in scored_p[:8]:
        if ov > 0:
            print(f"    {r['name']:30s}  ov={ov:.4f}  M={r['M_pred']:.4f} L={r['L']:.4f} D={r['D']:.4f}")

    print(f"\n  Using M_cohen:")
    scored_c = [(((r['M_cohen']*r['L']*r['D'])**(1/3) if r['M_cohen']>0 and r['L']>0 and r['D']>0 else
0), r)
                for r in results]
    scored_c.sort(key=lambda x: -x[0])
    for ov, r in scored_c[:8]:
        if ov > 0:
            print(f"    {r['name']:30s}  ov={ov:.4f}  M={r['M_cohen']:.4f} L={r['L']:.4f} D=
{r['D']:.4f}")


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