Python Code

```
"""
Activity Gate Sensitivity Test
==============================
Is the 1% threshold a magic number? Test L across a range of
thresholds to check stability of results.

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

import numpy as np
from collections import Counter

# ---- Minimal engines and emissions ----
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 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

# ---- L measure with variable threshold ----
def measure_L_with_threshold(E, threshold_pct, window=10):
    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, 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])

    if s_early < 1e-10: return 0.0, False
    reduction = (s_early - s_late) / s_early

    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 = (threshold_pct / 100.0) * 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, alive

# ---- Test systems ----
def main():
    print("=" * 75)
    print("  Activity Gate Threshold Sensitivity Test")
    print("  Does the 1% threshold matter?")
    print("=" * 75)

    # Systems that should be sensitive to the gate
    test_systems = {}

    # Systems that die (gate should catch them)
    test_systems['CA Rule 0 (dies)'] = emit_ca(run_ca(0, steps=300))
    test_systems['CA Rule 4 (static)'] = emit_ca(run_ca(4, steps=300))

    # Systems that nearly die (gate boundary)
    test_systems['GoL Diehard (dies at 130)'] = emit_gol(
        run_gol([[0,0,0,0,0,0,1,0],[1,1,0,0,0,0,0,0],[0,1,0,0,0,1,1,1]], steps=400))

    # Systems with clear learning (gate should not matter)
    test_systems['GoL R-pentomino'] = emit_gol(
        run_gol([[0,1,1],[1,1,0],[0,1,0]], steps=400))
    test_systems['CA Rule 54 (Class 4)'] = emit_ca(run_ca(54, steps=300))
    test_systems['CA Rule 30 (Class 3)'] = emit_ca(run_ca(30, steps=300))
    test_systems['CA Rule 184 (traffic)'] = emit_ca(run_ca(184, steps=300))

    # Gray-Scott
    print("  Generating Gray-Scott systems...")
    Uh, Vh = run_gs(0.028, 0.062, gs=64, steps=5000, sample=10)
    test_systems['GS Mitosis'] = emit_gs(Uh, Vh)
    Uh, Vh = run_gs(0.078, 0.061, gs=64, steps=5000, sample=10)
    test_systems['GS Death'] = emit_gs(Uh, Vh)
    Uh, Vh = run_gs(0.026, 0.051, gs=64, steps=5000, sample=10)
    test_systems['GS Chaos'] = emit_gs(Uh, Vh)

    # Thresholds to test
    thresholds = [0.001, 0.01, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0]

```

```
    print(f"\n  {'System':30s}", end="")
    for t in thresholds:
        print(f"  {t:>5.1f}%", end="")
    print("  | Gate flips?")
    print("  " + "-" * 100)

    for name, E in test_systems.items():
        print(f"  {name:30s}", end="")
        L_values = []
        alive_values = []
        for t in thresholds:
            L, alive = measure_L_with_threshold(E, t)
            L_values.append(L)
            alive_values.append(alive)
            print(f"  {L:5.3f}", end="")

        # Check if the gate ever flips
        if all(a == alive_values[0] for a in alive_values):
            flip = "No"
        else:
            # Find where it flips
            flip_at = None
            for i in range(1, len(alive_values)):
                if alive_values[i] != alive_values[i-1]:
                    flip_at = thresholds[i]
                    break
            flip = f"Yes, at {flip_at}%"

        print(f"  | {flip}")

    # Summary
    print(f"\n{'='*75}")
    print(f"  INTERPRETATION")
    print(f"{'='*75}")
    print("""
If the gate flips for a system between 0.1% and 5%, that system is
near the boundary and the threshold matters. If the gate never flips
(system is clearly alive or clearly dead), the threshold is irrelevant.

The question: how many systems change their L classification depending
on the threshold choice?
    """)

    # Now test ALL 256 CA rules at three thresholds
    print("  Testing all 256 CA rules at thresholds 0.1%, 1%, 5%...")

    changes = {'0.1_vs_1': 0, '1_vs_5': 0, '0.1_vs_5': 0}
    total_nonzero = 0

    for rule in range(256):
        grid = run_ca(rule, steps=300)
        E = emit_ca(grid)

        L_01, _ = measure_L_with_threshold(E, 0.1)
        L_1, _ = measure_L_with_threshold(E, 1.0)
        L_5, _ = measure_L_with_threshold(E, 5.0)

        if L_1 > 0:
            total_nonzero += 1

        if (L_01 > 0) != (L_1 > 0):
            changes['0.1_vs_1'] += 1
        if (L_1 > 0) != (L_5 > 0):
            changes['1_vs_5'] += 1
```

```
        if (L_01 > 0) != (L_5 > 0):
            changes['0.1_vs_5'] += 1

    print(f"\n  Of 256 CA rules ({total_nonzero} with L > 0 at 1% threshold):")
    print(f"    Rules that change L>0 status between 0.1% and 1%:  {changes['0.1_vs_1']}")
    print(f"    Rules that change L>0 status between 1% and 5%:    {changes['1_vs_5']}")
    print(f"    Rules that change L>0 status between 0.1% and 5%:  {changes['0.1_vs_5']}")

    pct = changes['0.1_vs_5'] / 256 * 100
    print(f"\n  Sensitivity: {pct:.1f}% of rules change classification across the full range")

    if pct < 5:
        print("  → ROBUST: threshold choice has minimal impact on results")
    elif pct < 15:
        print("  → MODERATELY SENSITIVE: some boundary cases affected")
    else:
        print("  → SENSITIVE: threshold choice significantly affects results")


if __name__ == '__main__':
    main()

```