Verification Code Dossier

Article 98

Quaternary Coding Theory of the Biological Ledger over GF(4)

Charles D. Schaper, Ph.D.

Volume 1, Number 4 November 2026

Certification Summary

The supplied standard-library Python program executes 110,516 explicit predicates in each of normal and optimized mode: 110,509 exact predicates and seven floating-point benchmark checks. Ten deliberately false alternatives are included as negative controls. Both modes produce byte-identical outputs. These are predicate executions, not a count of independent mathematical theorems.

1 What is independently checked

The suite constructs the field operations and all codewords, rather than using only the declared code parameters. It checks all 16 locus words, 64 hexamer words, 16 dual words and 4096 product words. The complete single-error decoder is tested on all 1024 received words and on all 64 by 16 transmitted-word/error pairs. The product code is explicitly generated from its six tensor rows; its minimum nonzero weight is nine, and every row and column has the claimed component-code membership.

For the $3 \times 3 \times 6$ box, the program explicitly constructs 54 vertices, 117 edges and 84 faces. Exact binary elimination gives ranks 53 and 64. These binary matrices have the same rank over GF(4), since extension of the scalar field preserves matrix rank. It checks every boundary-product entry, the girth lower bound and attaining square, and unit-capacity max-flow from a fixed root to every other vertex. The minimum flow is three. This proves that every nontrivial cut has size at least three; a degree-three corner attains it. Every nonzero error of weight at most two has a nonzero plaquette syndrome. The full CSS rank count is $117 - 53 - 64 = 0$. A counterexample shows that deleting a redundant displayed row need not reduce rank.

The exact failure polynomial is

$$p(\varepsilon) = 10\varepsilon^2 - 20\varepsilon^3 + 15\varepsilon^4 - 4\varepsilon^5.$$

It is checked against the binomial tail on 101 rational points. The general polynomial identity and its derivative factorization are elementary algebraic results in the manuscript; the finite checks do not replace those proofs. Maximum-entropy success is exactly 1/64. A correlated-error model need not obey this independent-contact polynomial.

2 Numerical benchmark conventions

At 300 K, the coded target $p = 0.01$ gives $\varepsilon = 0.0326821122$. Under the binary-logistic surrogate, the five-symbol uncoded comparison is 160.5043 to 87.5788 meV. With three equally degenerate wrong quaternary levels it is 188.9056 to 115.9801 meV. A fair three-symbol payload comparison instead starts at 147.2811 or 175.6824 meV, respectively. These are conditional equilibrium splitting

benchmarks, not measured molecular activation barriers, electric-field reductions, or net work savings.

3 Reproduction and provenance

Run `python run_verification.py`. It checks the SHA-256 source pin, executes normal and `-O` modes in separate temporary directories, compares all outputs, and saves the audit. No assertion is disabled by optimized mode: checks use an explicit exception-raising function. The source, macros and JSON report are included in the package. The complete listing below is byte-identical to the executed source.

The supplied historical Python script passed its own assertions. However, its product block computed parameter arithmetic, and its lattice block emitted rank/distance constants without constructing the full matrices. This revision adds the explicit computations described above. It does not establish a molecular encoder, non-destructive physical syndrome readout, quantum fault tolerance, or thermodynamic work reduction.

4 Complete executable source

```
#!/usr/bin/env python3
"""Article 98 explicit finite verification. Standard library only.
Counts are predicate executions, not independently proved theorems.
"""
import itertools as it, math, json
from fractions import Fraction as Q
from collections import Counter, deque
from pathlib import Path
counts=Counter()
def check(test, block, numeric=False):
    if not test: raise RuntimeError("FAILED: "+block)
    counts[block]+=1
    counts["numerical" if numeric else "exact"]+=1
F=range(4)
M=((0,0,0,0),(0,1,2,3),(0,2,3,1),(0,3,1,2))
def mul(a,b): return M[a][b]
def dot(a,b):
    v=0
    for x,y in zip(a,b): v ^= mul(x,y)
    return v
def wt(a): return sum(x!=0 for x in a)
def add(a,b): return tuple(x^y for x,y in zip(a,b))
def span(rows):
    return [tuple(dot(u,col) for col in zip(*rows))
            for u in it.product(F,repeat=len(rows))]
for a,b,c in it.product(F,repeat=3):
    check((a^b)^c==a^(b^c),'field')
    check(mul(mul(a,b),c)==mul(a,mul(b,c)),'field')
    check(mul(a,b^c)==mul(a,b)^mul(a,c),'field')
for a,b in it.product(F,repeat=2):
    check(mul(a,b)==mul(b,a) and (a^b)==(b^a),'field')
for a in F:
    check((a^a)==0 and mul(a,1)==a,'field')
    check(a==0 or any(mul(a,b)==1 for b in F),'field')
for a,b in it.product(range(2),repeat=2):
```

```
    check(b^((a^b)<<1)==mul(2,a^(b<<1)),'register')
# Additive projection of F2^3, not a field homomorphism.
for a,b in it.product(range(8),repeat=2):
    check(((a^b)&3)==((a&3)^(b&3)),'projection')
check(set(a&3 for a in range(8))==set(F),'projection')
G4=((1,1,1,1),(0,1,2,3)); C4=span(G4)
H5=((1,0,1,1,1),(0,1,1,2,3))
G5=((1,1,1,0,0),(1,2,0,1,0),(1,3,0,0,1))
C5=span(G5); D5=span(H5)
for code,size,dist,profile in [(C4,16,3,{0:1,3:12,4:3}),
 (C5,64,3,{0:1,3:30,4:15,5:18}),
 (D5,16,4,{0:1,4:15})]:
    check(len(set(code))==size,'small_codes')
    check(min(wt(x) for x in code if wt(x))==dist,'small_codes')
    check(dict(Counter(map(wt,code)))==profile,'small_codes')
    for a,b in it.product(code,repeat=2):
        check(add(a,b) in code,'small_codes')
for a,b in it.product(C4,repeat=2):check(dot(a,b)==0,'self_dual')
def syn5(y):return tuple(dot(h,y) for h in H5)
E5=[(0,)*5]+[tuple(a if j==i else 0 for j in range(5))
                    for i in range(5) for a in (1,2,3)]
lookup={syn5(e):e for e in E5}
check(len(lookup)==16,'decoder')
hits=Counter()
for y in it.product(F,repeat=5):
    e=lookup[syn5(y)]; c=add(y,e)
    check(c in C5 and wt(e)<=1,'decoder');hits[c]+=1
check(set(hits.values())=={16} and len(hits)==64,'decoder')
for c,e in it.product(C5,E5):
    y=add(c,e);check(add(y,lookup[syn5(y)])==c,'decoder')
# Explicit 20-coordinate tensor generator and all 4096 words.
GP=[tuple(mul(a,b) for a in x for b in y) for x in G4 for y in G5]
CP=span(GP); check(len(set(CP))==4096,'product')
check(min(wt(x) for x in CP if wt(x))==9,'product')
for c in CP:
    for r in range(4):check(c[5*r:5*r+5] in C5,'product')
    for j in range(5):check(tuple(c[5*r+j] for r in range(4)) in C4,'product')
# Explicit cubical boundaries, binary rows valid also over GF(4).
V=list(it.product(range(3),range(3),range(6))); vi={v:i for i,v in enumerate(V
    )}
E=[]
for v in V:
    for d in range(3):
        w=list(v);w[d]+=1;w=tuple(w)
        if w in vi:E.append((v,w))
ei={frozenset(e):i for i,e in enumerate(E)}
faces=[]
for v in V:
    for a,b in it.combinations(range(3),2):
        va=list(v);va[a]+=1;va=tuple(va)
        vb=list(v);vb[b]+=1;vb=tuple(vb)
        vab=list(va);vab[b]+=1;vab=tuple(vab)
        if va in vi and vb in vi and vab in vi:
            faces.append([ei[frozenset(e)] for e in [(v,va),(va,vab),(vab,vb)
                ,(vb,v)]])
B1=[sum(1<<j for j,e in enumerate(E) if v in e) for v in V]
B2=[sum(1<<j for j in f) for f in faces]
def rank(rows):
    piv={}
```

```
    for r in rows:
        while r:
            j=r.bit_length()-1
            if j in piv:r^=piv[j]
            else:piv[j]=r;break
    return len(piv)
check((len(V),len(E),len(faces))==(54,117,84),'lattice')
check((rank(B1),rank(B2))==(53,64),'lattice')
for a,b in it.product(B1,B2):check((a&b).bit_count()%2==0,'css')
check(117-rank(B1)-rank(B2)==0,'css')
for f in B2:check(f.bit_count()==4,'cycle')
# Girth = 4 by BFS with each edge removed.
adj={v:set() for v in V}
for a,b in E:adj[a].add(b);adj[b].add(a)
for a,b in E:
    q=deque([(a,0)]);seen={a};distance=None
    while q:
        x,d=q.popleft()
        if x==b:distance=d;break
        for y in adj[x]:
            if frozenset((x,y))==frozenset((a,b)):continue
            if y not in seen:seen.add(y);q.append((y,d+1))
    check(distance is not None and distance>=3,'cycle')
# Unit-capacity undirected max-flow from one root to every other vertex.
def flow(s,t):
    cap={(a,b):1 for a in V for b in adj[a]}; total=0
    while True:
        prev={s:None};q=deque([s])
        while q and t not in prev:
            a=q.popleft()
            for b in adj[a]:
                if cap[a,b]>0 and b not in prev:prev[b]=a;q.append(b)
        if t not in prev:return total
        b=t
        while b!=s:
            a=prev[b];cap[a,b]-=1;cap[b,a]+=1;b=a
        total+=1
flows=[flow(V[0],v) for v in V[1:]]
for f in flows:check(f>=3,'cut')
check(min(flows)==3 and min(map(len,adj.values()))==3,'cut')
# Every nonzero error of weight <=2 has nonzero plaquette syndrome.
columns=[sum(1<<i for i,b in enumerate(B2) if (b>>j)&1) for j in range(117)]
# Pair of binary masks represents the two GF(4) coefficients.
def scaled(mask,a):return (mask if a&1 else 0,mask if a&2 else 0)
seen={}
for j,a in it.product(range(117),(1,2,3)):
    s=scaled(columns[j],a);check(s!=(0,0) and s not in seen,'edge_errors');
      seen[s]=(j,a)
for i,j in it.combinations(range(117),2):
    for a,b in it.product((1,2,3),repeat=2):
        x,y=scaled(columns[i],a),scaled(columns[j],b)
        check((x[0]^y[0],x[1]^y[1])!=(0,0),'edge_errors')
# CSS counterexample: deleting a redundant displayed row need not lose rank.
check(any(rank(B2[:i]+B2[i+1:])==64 for i in range(84)),'css')
# Exact independent-contact failure polynomial and derivative factorization.
def fail(e):return 10*e**2-20*e**3+15*e**4-4*e**5
for k in range(101):
    e=Q(k,100)
    tail=sum(Q(math.comb(5,j))*e**j*(1-e)**(5-j) for j in range(2,6))
```

```
    check(fail(e)==tail,'probability')
    check(20*e-60*e**2+60*e**3-20*e**4==20*e*(1-e)**3,'probability')
# Floating benchmarks; no measured molecular energy or decoding work.
kb=8.617333262e-2;T=300
lo,hi=0.,1.
for _ in range(100):
    mid=(lo+hi)/2
    if fail(mid)<.01:lo=mid
    else:hi=mid
et=(lo+hi)/2
bench={'epsilon_coded_99':et}
for target in (.95,.99):
    e=1-target**.2;f=fail(e)
    bench[str(target)]={'epsilon':e,'coded_failure':f,'gain':(1-target)/f}
    check(abs((1-e)**5-target)<1e-12,'numeric',True)
check(abs(fail(et)-.01)<1e-12,'numeric',True)
for n in (3,5):
    e=1-.99**(1/n)
    for g in (1,3):
        key=f'n{n}_g{g}'
        u=kb*T*math.log(g*(1-e)/e);c=kb*T*math.log(g*(1-et)/et)
        bench[key]={'uncoded_meV':u,'coded_meV':c,'relative_reduction_pct'
            :100*(u-c)/u}
        check(c<u and u>0,'numeric',True)
# Exact full-noise success and correlated-error counterexample.
success=sum(correct==tuple([0]*5) for correct in
            (add(e,lookup[syn5(e)]) for e in it.product(F,repeat=5)))
check(Q(success,4**5)==Q(1,64),'probability')
check(Q(1,10)!=fail(Q(1,10)),'correlated_counterexample')
# Euclidean self-duality does not imply Hermitian self-orthogonality.
check(dot(G4[1],tuple(mul(x,x) for x in G4[1]))==1,'self_dual')
# Ten deliberately false alternatives rejected by the same predicates.
negative=[len(C4)==15,min(map(wt,C4[1:]))==4,
 len(C5)==32,len(lookup)==15,rank(B1)==54,rank(B2)==84,
 min(flows)==4,117-rank(B1)-rank(B2)==1,
 fail(Q(1,10))==Q(1,2),mul(2,2)==1]
for bad in negative:check(not bad,'negative_controls')
report={'checks_total':counts['exact']+counts['numerical'],
 'checks_exact':counts['exact'],'checks_numerical':counts['numerical'],
 'negative_controls_included':10,'blocks':dict(counts),'benchmarks':bench,
 'scope':'Finite field, code, product enumeration, cubical matrices, graph
     cuts, decoder and independent-error model; no molecular validation.'}
macros={
 'valGFFourSize':4,'valLocusLen':4,'valLocusDim':2,'valLocusDist':3,
 'valLocusCodewords':16,'valLocusWtThree':12,'valLocusWtFour':3,
 'valHexLen':5,'valHexDim':3,'valHexDist':3,'valHexCodewords':64,
 'valHexSphereSize':16,'valHexSpaceSize':'1,024','valHexSyndromes':15,
 'valHexWtThree':30,'valHexWtFour':15,'valHexWtFive':18,
 'valHexDualDim':2,'valHexDualDist':4,'valHexDualWtFour':15,
 'valProdLen':20,'valProdDim':6,'valProdDist':9,'valProdCorr':4,
 'valProdCodewords':'4,096','valLatVertices':len(V),'valLatEdges':len(E),
 'valLatPlaquettes':len(faces),'valCycDim':rank(B2),'valCycDist':4,
 'valCutDim':rank(B1),'valCutDist':min(flows),
 'valFailUncNinetyFive':'0.0500','valFailUncNinetyNine':'0.0100',
 'valFailCodNinetyFive':format(bench['0.95']['coded_failure'],'.6f'),
 'valFailCodNinetyNine':format(bench['0.99']['coded_failure'],'.7f'),
 'valGainNinetyFive':format(bench['0.95']['gain'],'.1f'),
 'valGainNinetyNine':format(bench['0.99']['gain'],'.1f'),
 'valStarkUncMeV':format(bench['n5_g1']['uncoded_meV'],'.2f'),
```

```
 'valStarkCodMeV':format(bench['n5_g1']['coded_meV'],'.2f'),
 'valStarkSavingsPct':format(bench['n5_g1']['relative_reduction_pct'],'.1f')}
Path('verification_values_a98.tex').write_text(
 '% Generated by executed Article 98 verification source.\n'+
 ''.join('\\newcommand{\\'+k+'}{'+str(v)+'}\n' for k,v in macros.items()))
Path('verification_report.json').write_text(json.dumps(report,indent=2,
    sort_keys=True)+'\n')
print(json.dumps(report,sort_keys=True))
```