

PRD 13: Higher-Order Topological Insulators from Crystalline Symmetry

Pure Thought AI Challenge 13

Pure Thought AI Challenges Project

January 18, 2026

Abstract

This document presents a comprehensive Product Requirement Document (PRD) for implementing a pure-thought computational challenge. The problem can be tackled using only symbolic mathematics, exact arithmetic, and fresh code—no experimental data or materials databases required until final verification. All results must be accompanied by machine-checkable certificates.

Contents

Domain: Materials Science

Timeline: 5-8 months

Difficulty: High

Prerequisites: Topological band theory, crystalline symmetry, representation theory, K-theory

0.1 1. Problem Statement

0.1.1 Scientific Context

Higher-order topological insulators (HOTIs) extend topological band theory beyond conventional wisdom:

- **1st-order TI:** (d-1)-dimensional edge states on d-dimensional bulk (e.g., 1D edge states in 2D)
- **2nd-order TI:** (d-2)-dimensional corner/hinge states (e.g., 0D corner states in 2D)
- **nth-order TI:** (d-n)-dimensional boundary states

The key insight: **crystalline symmetries** (rotation, mirror, inversion) protect higher-order topology even when time-reversal/particle-hole symmetries are absent.

Examples:

- **2D Quadrupole Insulator:** Square lattice with C rotation → corner charges quantized to $\pm e/2$
- **3D Hinge Insulator:** Cubic lattice with mirror symmetries → 1D hinge modes on edges
- **Breathing Kagome:** Corner states protected by C rotation

Bulk-Boundary Correspondence: Traditional correspondence (Chern number → edge modes) fails for HOTIs. New invariants needed:

- **Nested Wilson loops:** Multipole moments (dipole, quadrupole, octupole)
- **Symmetry indicators:** Irrep decomposition at high-symmetry points
- **Corner charge formula:** $Q_{corner} = e(P_x P_y - P_x - P_y) mode$

0.1.2 Core Question

Can we systematically construct tight-binding models with higher-order topology using ONLY crystalline symmetry and representation theory—without trial-and-error or simulations?

Specifically:

- Given space group G and target (corner charge Q_c , hinge modes N_h), construct Hamiltonian
- Prove corner/hinge states exist using nested Wilson loops
- Compute multipole moments exactly (rational arithmetic)
- Certify robustness against symmetry-preserving disorder
- Classify all possible HOTIs for 2D wallpaper groups and 3D space groups

0.1.3 Why This Matters

Theoretical Impact:

- Completes classification of topological phases beyond Altland-Zirnbauer
- Connects topology to crystallography and group cohomology
- Reveals new bulk-boundary principles

Practical Benefits:

- Design materials with fractional charges at corners
- Quantum information: corner states as protected qubits
- Sensing: corner modes concentrate electromagnetic fields

Pure Thought Advantages:

- Multipole moments are purely algebraic (Wilson loop eigenvalues)
- Symmetry indicators computed from irreps (character tables)
- No material data needed—geometry + symmetry suffice
- Exact classification possible via K-theory

0.2 2. Mathematical Formulation

0.2.1 Problem Definition

A **higher-order topological insulator** (HOTI) is a Hamiltonian $H(k)$ with:

- **Bulk Gap:** No states at Fermi energy in bulk
- **Gapped Edges:** $(d-1)$ -dimensional boundaries also gapped
- **Corner/Hinge States:** Localized $(d-n)$ -dimensional modes at n -codimension boundaries

Quadrupole Moment (2D):

$$1 \quad q_{xy} = (1/2) \int_{\text{BZ}} \text{Tr} [P (\hat{x} P \hat{y} P - \hat{y} P \hat{x} P)] dk$$

Nested Wilson Loop:

$$\begin{aligned} 1 \quad W_x(k_y) &= \exp(i \int_{-k_y}^{0} \{0\}^2 A_x(k_x, k_y) dk_x) \\ 2 \quad W_y(k_y) &= \text{eigenphases of } W_x(k_y) \\ 3 \quad W_y &= \exp(i \int_{-k_y}^{0} \{0\}^2 A_y(k_y) dk_y) \end{aligned}$$

Eigenphases of W_y give quantized polarization & quadrupole moment.

Corner Charge Formula:

$$1 \quad Q_{corner} = e(p_x p_y - p_x - p_y) \bmod e$$

where $p_x, p_y \in \{0, 1/2\}$ are bulk polarizations.

Symmetry Indicator (for space group G):

$$1 \quad z = (n_n, n_X, n_M, n_Y) \bmod 2$$

where $n_K = (\text{number of occupied bands with specific irrep } K) \bmod 2$.

0.2.2 Certificate Requirements

- **Multipole Certificate:** Exact quadrupole/octupole moment (rational number)
- **Corner State Count:** Number of zero-energy corner modes
- **Nested Wilson Loop Spectrum:** Eigenphases $i(k)$
- **Symmetry Indicator:** Irrep content at all high-symmetry points
- **Robustness Proof:** Corner states survive disorder preserving crystalline symmetry

0.2.3 Input/Output Specification

Input:

```

1 from sympy import *
2 import numpy as np
3 from typing import List, Callable, Tuple
4
5 class CrystallineHamiltonian:
6     dimension: int # 2D or 3D
7     space_group: int # International number
8     point_group: str # Schoenflies notation (C4v, D4h, etc.)
9
10    hamiltonian: Callable[[np.ndarray, np.ndarray]] # H(k)
11    filling: int # Number of occupied bands
12
13    symmetry_operators: dict # {name: unitary matrix} for C4, mirror,
        etc.

```

Output:

```

1 class HOTICertificate:
2     model: CrystallineHamiltonian
3
4     # Topology
5     quadrupole_moment: Fraction # q_xy {0, 1/2} for 2D
6     octupole_moment: Optional[Fraction] # For 3D
7
8     nested_wilson_spectrum: List[List[float]] # -i ^ (k_ )
9     bulk_polarizations: Tuple[Fraction, Fraction] # (p_x, p_y)
10
11    # Symmetry analysis
12    symmetry_indicator: Tuple[int, ...] # (n_ , n_X, ...) mod 2
13    irrep_decomposition: dict # Irrep content at each high-sym point
14
15    # Corner/hinge states
16    corner_states: List[np.ndarray] # Wavefunctions localized to
        corners
17    corner_charges: List[Fraction] # Charge at each corner
18    hinge_dispersion: Optional[np.ndarray] # For 3D systems
19
20    # Verification
21    bulk_gap: float
22    edge_gap: float # Confirms edges are gapped
23    localization_length: float # Corner state decay into bulk

```

```
24     proof_of_quantization: str  # Derivation showing q_xy      {0, 1/2}
25 
```

0.3 3. Implementation Approach

0.3.1 Phase 1: Benalcazar-Bernevig-Hughes Model (Months 1-2)

Implement canonical 2D quadrupole insulator:

```
1 import numpy as np
2 from sympy import *
3 from scipy.linalg import eigh
4
5 def bbh_model(gamma: float, lambda_param: float) -> Callable:
6     """
7         Benalcazar-Bernevig-Hughes (BBH) quadrupole insulator.
8
9     2D square lattice with 4 orbitals per site.
10    C4 rotation symmetry protects corner charges e /2.
11
12    Parameters:
13        - gamma: intracell hopping (0 < gamma < 1)
14        - lambda_param: intercell hopping (0 < lambda < 1)
15
16    For gamma > lambda: trivial
17    For gamma < lambda: topological (q_xy = 1/2)
18    """
19
20    def H(k: np.ndarray) -> np.ndarray:
21        kx, ky = k[0], k[1]
22
23        # Pauli matrices for sublattice
24        sx = np.array([[0, 1], [1, 0]])
25        sy = np.array([[0, -1j], [1j, 0]])
26        sz = np.array([[1, 0], [0, -1]])
27        s0 = np.eye(2)
28
29        # Hamiltonian (4 4 = 2 orbitals 2 sublattices)
30        H_k = (
31            (gamma + lambda_param * np.cos(kx)) * np.kron(sx, s0) +
32            (gamma + lambda_param * np.cos(ky)) * np.kron(sy, s0) +
33            lambda_param * np.sin(kx) * np.kron(sz, sx) +
34            lambda_param * np.sin(ky) * np.kron(sz, sy)
35        )
36
37        return H_k
38
39    return H
40
41    def verify_c4_symmetry(H_func: Callable) -> bool:
42        """
43            Verify Hamiltonian has C4 rotation symmetry.
44
45            C4: (kx, ky)      (-ky, kx)
46            H(C4 k) = U_C4 H(k) U_C4
47 
```

```

46 """
47 # C4 operator (90    rotation in orbital space)
48 U_C4 = np.array([
49     [0, 0, 0, 1],
50     [1, 0, 0, 0],
51     [0, 1, 0, 0],
52     [0, 0, 1, 0]
53 ]) # Cyclic permutation
54
55 # Test at random k-points
56 for _ in range(10):
57     k = np.random.uniform(-np.pi, np.pi, 2)
58     k_rot = np.array([-k[1], k[0]]) # C4 rotation in k-space
59
60     H_k = H_func(k)
61     H_k_rot = H_func(k_rot)
62
63     # Check symmetry relation
64     lhs = H_k_rot
65     rhs = U_C4 @ H_k @ U_C4.conj().T
66
67     if not np.allclose(lhs, rhs, atol=1e-10):
68         return False
69
70 return True

```

Validation: Reproduce BBH phase diagram (gamma vs lambda), verify corner charges.

0.3.2 Phase 2: Nested Wilson Loops (Months 2-4)

Compute multipole moments via nested Wilson loops:

```

1 def wilson_loop_x(H_func: Callable, ky: float, band_indices: List[int],
2                     N_kx: int = 100) -> np.ndarray:
3     """
4     Compute Wilson loop in x-direction at fixed ky.
5
6     W_x(ky) = exp(i      A_x(kx, ky) dkx)
7
8     Returns: Unitary matrix W_x
9     """
10    kx_values = np.linspace(0, 2*np.pi, N_kx, endpoint=False)
11    dk_x = kx_values[1] - kx_values[0]
12
13    # Initialize Wilson loop as identity
14    W_x = np.eye(len(band_indices), dtype=complex)
15
16    for i, kx in enumerate(kx_values):
17        k = np.array([kx, ky])
18        k_next = np.array([(kx + dk_x) % (2*np.pi), ky])
19
20        # Get occupied states at k and k+dk
21        evals, evecs = eigh(H_func(k))
22        sorted_idx = np.argsort(evals)
23        states_k = evecs[:, sorted_idx[band_indices]]
24

```

```

25     evals_next, evecs_next = eigh(H_func(k_next))
26     sorted_idx_next = np.argsort(evals_next)
27     states_k_next = evecs_next[:, sorted_idx_next[band_indices]]
28
29     # Overlap matrix
30     F = states_k.conj().T @ states_k_next
31
32     # Update Wilson loop
33     W_x = W_x @ F
34
35     return W_x
36
37 def nested_wilson_loop(H_func: Callable, band_indices: List[int],
38                         N_kx: int = 100, N_ky: int = 100) -> np.ndarray:
39     """
40         Compute nested Wilson loop to extract quadrupole moment.
41
42         1. Compute W_x(ky) for each ky
43         2. Diagonalize to get eigenphases _i (ky)
44         3. Compute Wilson loop of _i (ky) in ky-direction
45         4. Final eigenphases give quantized polarization
46     """
47     ky_values = np.linspace(0, 2*np.pi, N_ky, endpoint=False)
48
49     # Array to store eigenphases _i (ky)
50     nu_spectrum = np.zeros((N_ky, len(band_indices)))
51
52     for j, ky in enumerate(ky_values):
53         W_x = wilson_loop_x(H_func, ky, band_indices, N_kx)
54
55         # Eigenvalues of W_x = exp(i _i )
56         eigenvalues = np.linalg.eigvals(W_x)
57         nu_values = np.angle(eigenvalues) # Phases [- , ]
58
59         nu_spectrum[j, :] = np.sort(nu_values)
60
61     # Now compute Wilson loop in y-direction using _i (ky)
62     # This is tricky need to track which belongs to which band
63
64     # Simplified: compute winding of each _i (ky)
65     polarizations = []
66
67     for i in range(len(band_indices)):
68         # Winding number of _i (ky)
69         nu_traj = nu_spectrum[:, i]
70
71         # Total phase accumulated (account for 2 jumps)
72         total_phase = np.sum(np.diff(np.unwrap(nu_traj)))
73         p_i = total_phase / (2*np.pi)
74
75         polarizations.append(p_i)
76
77     return polarizations, nu_spectrum
78
79 def compute_quadrupole_moment(H_func: Callable, band_indices:
80     List[int]) -> Fraction:

```

```

80 """
81     Compute quantized quadrupole moment q_xy.
82
83     q_xy = (p_x p_y - p_x - p_y) / 2 mod 1/2
84
85     where p_x, p_y are Wannier center polarizations.
86 """
87
88 # Compute nested Wilson loops in both directions
89 p_x_list, _ = nested_wilson_loop(H_func, band_indices)
90 p_y_list, _ = nested_wilson_loop(H_func, band_indices) # Need to
91     swap directions
92
93 # For filled bands, take sum of polarizations mod 1
94 p_x = sum(p_x_list) % 1
95 p_y = sum(p_y_list) % 1
96
97 # Quadrupole formula
98 q_xy = (p_x * p_y - p_x - p_y) / 2
99
100 # Quantize to {0, 1/2}
101 if abs(q_xy) < 0.25:
102     return Fraction(0, 1)
103 elif abs(q_xy - 0.5) < 0.25 or abs(q_xy + 0.5) < 0.25:
104     return Fraction(1, 2)
105 else:
106     # Should not happen for topological systems
107     return Fraction(int(round(2*q_xy)), 2)

```

0.3.3 Phase 3: Corner State Calculation (Months 4-5)

Solve for corner-localized modes in finite geometry:

```

1 def finite_lattice_hamiltonian(H_bulk: Callable, L_x: int, L_y: int) ->
2     np.ndarray:
3         """
4             Construct Hamiltonian for finite L_x      L_y lattice with open
5                 boundaries.
6
7             Each unit cell has N_orb orbitals.
8             Total Hilbert space dimension: N_orb      L_x      L_y
9         """
10
11         # Get unit cell Hamiltonian dimension
12         H_test = H_bulk(np.array([0, 0]))
13         N_orb = H_test.shape[0]
14
15         dim = N_orb * L_x * L_y
16         H_finite = np.zeros((dim, dim), dtype=complex)
17
18         for ix in range(L_x):
19             for iy in range(L_y):
20                 # On-site terms
21                 idx = (ix * L_y + iy) * N_orb
22
23                     # Intracell Hamiltonian (k=0 term)
24                     H_00 = H_bulk(np.array([0, 0]))
25                     H_finite[idx:idx+N_orb, idx:idx+N_orb] = H_00

```

```

23
24     # Hopping in x-direction
25     if ix < L_x - 1:
26         idx_next_x = ((ix+1) * L_y + iy) * N_orb
27
28     # Extract hopping from k-dependence
29     H_kx = H_bulk(np.array([np.pi/L_x, 0])) # Small kx
30     t_x = (H_kx - H_00) / (1j * np.pi/L_x) # Linear term
31
32     H_finite[idx:idx+N_orb, idx_next_x:idx_next_x+N_orb] =
33         t_x
34     H_finite[idx_next_x:idx_next_x+N_orb, idx:idx+N_orb] =
35         t_x.conj().T
36
37     # Hopping in y-direction
38     if iy < L_y - 1:
39         idx_next_y = (ix * L_y + (iy+1)) * N_orb
40
41         H_ky = H_bulk(np.array([0, np.pi/L_y]))
42         t_y = (H_ky - H_00) / (1j * np.pi/L_y)
43
44         H_finite[idx:idx+N_orb, idx_next_y:idx_next_y+N_orb] =
45             t_y
46         H_finite[idx_next_y:idx_next_y+N_orb, idx:idx+N_orb] =
47             t_y.conj().T
48
49     return H_finite
50
51
52 def find_corner_states(H_bulk: Callable, L_x: int = 20, L_y: int = 20,
53                         energy_threshold: float = 0.01) ->
54     List[np.ndarray]:
55
56     """
57     Find in-gap corner states for finite system.
58     """
59
60     H_finite = finite_lattice_hamiltonian(H_bulk, L_x, L_y)
61
62     # Diagonalize
63     eigenvalues, eigenvectors = eigh(H_finite)
64
65     # Find states near zero energy (in gap)
66     gap_indices = np.where(np.abs(eigenvalues) < energy_threshold)[0]
67
68     corner_states = [eigenvectors[:, idx] for idx in gap_indices]
69
70     return corner_states, eigenvalues[gap_indices]
71
72 def compute_corner_charge(corner_state: np.ndarray, L_x: int, L_y: int,
73                           N_orb: int) -> Fraction:
74
75     """
76     Compute charge localized at corner.
77
78     Integrate | | in corner region (e.g., 5 5 sites around corner).
79     """
80
81     # Reshape wavefunction to lattice
82     psi_lattice = corner_state.reshape((L_x, L_y, N_orb))

```

```

74 # Define corner region (bottom-left as example)
75 corner_size = min(5, L_x//4, L_y//4)
76
77 corner_charge = 0
78 for ix in range(corner_size):
79     for iy in range(corner_size):
80         # Sum over orbitals
81         corner_charge += np.sum(np.abs(psi_lattice[ix, iy, :])**2)
82
83 # Quantize (should be 1/2 for HOTI)
84 if abs(corner_charge - 0.5) < 0.1:
85     return Fraction(1, 2)
86 elif abs(corner_charge) < 0.1:
87     return Fraction(0, 1)
88 else:
89     return Fraction(int(round(2*corner_charge)), 2)

```

0.3.4 Phase 4: Symmetry Indicators (Months 5-6)

Compute irrep decomposition at high-symmetry points:

```

1 def compute_symmetry_indicator(H_func: Callable, space_group: int,
2                                 band_indices: List[int]) -> Tuple[int,
3                                     ...]:
4     """
5     Compute symmetry indicator z = (n_ , n_X, n_M, n_Y) mod 2.
6
7     For each high-symmetry point K, count occupied bands with specific
8     irreps.
9     """
10
11    # Get high-symmetry points for space group
12    high_sym_points = get_high_symmetry_points_2d(space_group)
13
14    indicators = []
15
16    for K_name, k_point in high_sym_points:
17        H_K = H_func(k_point)
18        evals, evecs = eigh(H_K)
19
20        # Get occupied states
21        sorted_idx = np.argsort(evals)
22        occupied_states = evecs[:, sorted_idx[band_indices]]
23
24        # Determine irrep content using character table
25        irrep_counts = decompose_into_irreps(H_K, occupied_states,
26                                              k_point, space_group)
27
28        # Specific indicator: e.g., number of A1g reps mod 2
29        n_K = irrep_counts['A1g'] % 2 # Convention depends on space
30                                     group
31
32        indicators.append(n_K)
33
34    return tuple(indicators)
35
36 def decompose_into_irreps(H_K: np.ndarray, states: np.ndarray,
37                           character_table: dict):
38
39    irreps = {key: value for key, value in character_table.items()
40              if len(value) == H_K.shape[0]}
41
42    irreps["E"] = irreps["E1"] + irreps["E2"]
43
44    irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
45
46    irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
47
48    irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
49
50    irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
51
52    irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
53
54    irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
55
56    irreps["A1g"] = np.sum(states * irreps["A1g"], axis=1)
57
58    irreps["Eg"] = np.sum(states * irreps["Eg"], axis=1)
59
60    irreps["T1g"] = np.sum(states * irreps["T1g"], axis=1)
61
62    irreps["T2g"] = np.sum(states * irreps["T2g"], axis=1)
63
64    irreps["T1u"] = np.sum(states * irreps["T1u"], axis=1)
65
66    irreps["T2u"] = np.sum(states * irreps["T2u"], axis=1)
67
68    irreps["A1u"] = np.sum(states * irreps["A1u"], axis=1)
69
70    irreps["Eg"] = irreps["Eg"] / 2
71
72    irreps["T1g"] = irreps["T1g"] / 2
73
74    irreps["T2g"] = irreps["T2g"] / 2
75
76    irreps["T1u"] = irreps["T1u"] / 2
77
78    irreps["T2u"] = irreps["T2u"] / 2
79
80    irreps["A1u"] = irreps["A1u"] / 2
81
82    irreps["E"] = irreps["E1"] + irreps["E2"]
83
84    irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
85
86    irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
87
88    irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
89
90    irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
91
92    irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
93
94    irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
95
96    irreps["A1g"] = irreps["A1g"] / 2
97
98    irreps["Eg"] = irreps["Eg"] / 2
99
100   irreps["T1g"] = irreps["T1g"] / 2
101
102   irreps["T2g"] = irreps["T2g"] / 2
103
104   irreps["T1u"] = irreps["T1u"] / 2
105
106   irreps["T2u"] = irreps["T2u"] / 2
107
108   irreps["A1u"] = irreps["A1u"] / 2
109
110   irreps["E"] = irreps["E1"] + irreps["E2"]
111
112   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
113
114   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
115
116   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
117
118   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
119
120   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
121
122   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
123
124   irreps["A1g"] = irreps["A1g"] / 2
125
126   irreps["Eg"] = irreps["Eg"] / 2
127
128   irreps["T1g"] = irreps["T1g"] / 2
129
130   irreps["T2g"] = irreps["T2g"] / 2
131
132   irreps["T1u"] = irreps["T1u"] / 2
133
134   irreps["T2u"] = irreps["T2u"] / 2
135
136   irreps["A1u"] = irreps["A1u"] / 2
137
138   irreps["E"] = irreps["E1"] + irreps["E2"]
139
140   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
141
142   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
143
144   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
145
146   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
147
148   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
149
150   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
151
152   irreps["A1g"] = irreps["A1g"] / 2
153
154   irreps["Eg"] = irreps["Eg"] / 2
155
156   irreps["T1g"] = irreps["T1g"] / 2
157
158   irreps["T2g"] = irreps["T2g"] / 2
159
160   irreps["T1u"] = irreps["T1u"] / 2
161
162   irreps["T2u"] = irreps["T2u"] / 2
163
164   irreps["A1u"] = irreps["A1u"] / 2
165
166   irreps["E"] = irreps["E1"] + irreps["E2"]
167
168   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
169
170   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
171
172   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
173
174   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
175
176   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
177
178   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
179
180   irreps["A1g"] = irreps["A1g"] / 2
181
182   irreps["Eg"] = irreps["Eg"] / 2
183
184   irreps["T1g"] = irreps["T1g"] / 2
185
186   irreps["T2g"] = irreps["T2g"] / 2
187
188   irreps["T1u"] = irreps["T1u"] / 2
189
190   irreps["T2u"] = irreps["T2u"] / 2
191
192   irreps["A1u"] = irreps["A1u"] / 2
193
194   irreps["E"] = irreps["E1"] + irreps["E2"]
195
196   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
197
198   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
199
200   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
201
202   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
203
204   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
205
206   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
207
208   irreps["A1g"] = irreps["A1g"] / 2
209
210   irreps["Eg"] = irreps["Eg"] / 2
211
212   irreps["T1g"] = irreps["T1g"] / 2
213
214   irreps["T2g"] = irreps["T2g"] / 2
215
216   irreps["T1u"] = irreps["T1u"] / 2
217
218   irreps["T2u"] = irreps["T2u"] / 2
219
220   irreps["A1u"] = irreps["A1u"] / 2
221
222   irreps["E"] = irreps["E1"] + irreps["E2"]
223
224   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
225
226   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
227
228   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
229
230   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
231
232   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
233
234   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
235
236   irreps["A1g"] = irreps["A1g"] / 2
237
238   irreps["Eg"] = irreps["Eg"] / 2
239
240   irreps["T1g"] = irreps["T1g"] / 2
241
242   irreps["T2g"] = irreps["T2g"] / 2
243
244   irreps["T1u"] = irreps["T1u"] / 2
245
246   irreps["T2u"] = irreps["T2u"] / 2
247
248   irreps["A1u"] = irreps["A1u"] / 2
249
250   irreps["E"] = irreps["E1"] + irreps["E2"]
251
252   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
253
254   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
255
256   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
257
258   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
259
260   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
261
262   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
263
264   irreps["A1g"] = irreps["A1g"] / 2
265
266   irreps["Eg"] = irreps["Eg"] / 2
267
268   irreps["T1g"] = irreps["T1g"] / 2
269
270   irreps["T2g"] = irreps["T2g"] / 2
271
272   irreps["T1u"] = irreps["T1u"] / 2
273
274   irreps["T2u"] = irreps["T2u"] / 2
275
276   irreps["A1u"] = irreps["A1u"] / 2
277
278   irreps["E"] = irreps["E1"] + irreps["E2"]
279
280   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
281
282   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
283
284   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
285
286   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
287
288   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
289
290   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
291
292   irreps["A1g"] = irreps["A1g"] / 2
293
294   irreps["Eg"] = irreps["Eg"] / 2
295
296   irreps["T1g"] = irreps["T1g"] / 2
297
298   irreps["T2g"] = irreps["T2g"] / 2
299
300   irreps["T1u"] = irreps["T1u"] / 2
301
302   irreps["T2u"] = irreps["T2u"] / 2
303
304   irreps["A1u"] = irreps["A1u"] / 2
305
306   irreps["E"] = irreps["E1"] + irreps["E2"]
307
308   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
309
310   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
311
312   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
313
314   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
315
316   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
317
318   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
319
320   irreps["A1g"] = irreps["A1g"] / 2
321
322   irreps["Eg"] = irreps["Eg"] / 2
323
324   irreps["T1g"] = irreps["T1g"] / 2
325
326   irreps["T2g"] = irreps["T2g"] / 2
327
328   irreps["T1u"] = irreps["T1u"] / 2
329
330   irreps["T2u"] = irreps["T2u"] / 2
331
332   irreps["A1u"] = irreps["A1u"] / 2
333
334   irreps["E"] = irreps["E1"] + irreps["E2"]
335
336   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
337
338   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
339
340   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
341
342   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
343
344   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
345
346   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
347
348   irreps["A1g"] = irreps["A1g"] / 2
349
350   irreps["Eg"] = irreps["Eg"] / 2
351
352   irreps["T1g"] = irreps["T1g"] / 2
353
354   irreps["T2g"] = irreps["T2g"] / 2
355
356   irreps["T1u"] = irreps["T1u"] / 2
357
358   irreps["T2u"] = irreps["T2u"] / 2
359
360   irreps["A1u"] = irreps["A1u"] / 2
361
362   irreps["E"] = irreps["E1"] + irreps["E2"]
363
364   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
365
366   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
367
368   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
369
370   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
371
372   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
373
374   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
375
376   irreps["A1g"] = irreps["A1g"] / 2
377
378   irreps["Eg"] = irreps["Eg"] / 2
379
380   irreps["T1g"] = irreps["T1g"] / 2
381
382   irreps["T2g"] = irreps["T2g"] / 2
383
384   irreps["T1u"] = irreps["T1u"] / 2
385
386   irreps["T2u"] = irreps["T2u"] / 2
387
388   irreps["A1u"] = irreps["A1u"] / 2
389
390   irreps["E"] = irreps["E1"] + irreps["E2"]
391
392   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
393
394   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
395
396   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
397
398   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
399
400   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
401
402   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
403
404   irreps["A1g"] = irreps["A1g"] / 2
405
406   irreps["Eg"] = irreps["Eg"] / 2
407
408   irreps["T1g"] = irreps["T1g"] / 2
409
410   irreps["T2g"] = irreps["T2g"] / 2
411
412   irreps["T1u"] = irreps["T1u"] / 2
413
414   irreps["T2u"] = irreps["T2u"] / 2
415
416   irreps["A1u"] = irreps["A1u"] / 2
417
418   irreps["E"] = irreps["E1"] + irreps["E2"]
419
420   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
421
422   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
423
424   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
425
426   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
427
428   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
429
430   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
431
432   irreps["A1g"] = irreps["A1g"] / 2
433
434   irreps["Eg"] = irreps["Eg"] / 2
435
436   irreps["T1g"] = irreps["T1g"] / 2
437
438   irreps["T2g"] = irreps["T2g"] / 2
439
440   irreps["T1u"] = irreps["T1u"] / 2
441
442   irreps["T2u"] = irreps["T2u"] / 2
443
444   irreps["A1u"] = irreps["A1u"] / 2
445
446   irreps["E"] = irreps["E1"] + irreps["E2"]
447
448   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
449
450   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
451
452   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
453
454   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
455
456   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
457
458   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
459
460   irreps["A1g"] = irreps["A1g"] / 2
461
462   irreps["Eg"] = irreps["Eg"] / 2
463
464   irreps["T1g"] = irreps["T1g"] / 2
465
466   irreps["T2g"] = irreps["T2g"] / 2
467
468   irreps["T1u"] = irreps["T1u"] / 2
469
470   irreps["T2u"] = irreps["T2u"] / 2
471
472   irreps["A1u"] = irreps["A1u"] / 2
473
474   irreps["E"] = irreps["E1"] + irreps["E2"]
475
476   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
477
478   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
479
480   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
481
482   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
483
484   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
485
486   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
487
488   irreps["A1g"] = irreps["A1g"] / 2
489
490   irreps["Eg"] = irreps["Eg"] / 2
491
492   irreps["T1g"] = irreps["T1g"] / 2
493
494   irreps["T2g"] = irreps["T2g"] / 2
495
496   irreps["T1u"] = irreps["T1u"] / 2
497
498   irreps["T2u"] = irreps["T2u"] / 2
499
500   irreps["A1u"] = irreps["A1u"] / 2
501
502   irreps["E"] = irreps["E1"] + irreps["E2"]
503
504   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
505
506   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
507
508   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
509
510   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
511
512   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
513
514   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
515
516   irreps["A1g"] = irreps["A1g"] / 2
517
518   irreps["Eg"] = irreps["Eg"] / 2
519
520   irreps["T1g"] = irreps["T1g"] / 2
521
522   irreps["T2g"] = irreps["T2g"] / 2
523
524   irreps["T1u"] = irreps["T1u"] / 2
525
526   irreps["T2u"] = irreps["T2u"] / 2
527
528   irreps["A1u"] = irreps["A1u"] / 2
529
530   irreps["E"] = irreps["E1"] + irreps["E2"]
531
532   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
533
534   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
535
536   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
537
538   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
539
540   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
541
542   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
543
544   irreps["A1g"] = irreps["A1g"] / 2
545
546   irreps["Eg"] = irreps["Eg"] / 2
547
548   irreps["T1g"] = irreps["T1g"] / 2
549
550   irreps["T2g"] = irreps["T2g"] / 2
551
552   irreps["T1u"] = irreps["T1u"] / 2
553
554   irreps["T2u"] = irreps["T2u"] / 2
555
556   irreps["A1u"] = irreps["A1u"] / 2
557
558   irreps["E"] = irreps["E1"] + irreps["E2"]
559
560   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
561
562   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
563
564   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
565
566   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
567
568   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
569
570   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
571
572   irreps["A1g"] = irreps["A1g"] / 2
573
574   irreps["Eg"] = irreps["Eg"] / 2
575
576   irreps["T1g"] = irreps["T1g"] / 2
577
578   irreps["T2g"] = irreps["T2g"] / 2
579
580   irreps["T1u"] = irreps["T1u"] / 2
581
582   irreps["T2u"] = irreps["T2u"] / 2
583
584   irreps["A1u"] = irreps["A1u"] / 2
585
586   irreps["E"] = irreps["E1"] + irreps["E2"]
587
588   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
589
590   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
591
592   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
593
594   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
595
596   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
597
598   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
599
600   irreps["A1g"] = irreps["A1g"] / 2
601
602   irreps["Eg"] = irreps["Eg"] / 2
603
604   irreps["T1g"] = irreps["T1g"] / 2
605
606   irreps["T2g"] = irreps["T2g"] / 2
607
608   irreps["T1u"] = irreps["T1u"] / 2
609
610   irreps["T2u"] = irreps["T2u"] / 2
611
612   irreps["A1u"] = irreps["A1u"] / 2
613
614   irreps["E"] = irreps["E1"] + irreps["E2"]
615
616   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
617
618   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
619
620   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
621
622   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
623
624   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
625
626   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
627
628   irreps["A1g"] = irreps["A1g"] / 2
629
630   irreps["Eg"] = irreps["Eg"] / 2
631
632   irreps["T1g"] = irreps["T1g"] / 2
633
634   irreps["T2g"] = irreps["T2g"] / 2
635
636   irreps["T1u"] = irreps["T1u"] / 2
637
638   irreps["T2u"] = irreps["T2u"] / 2
639
640   irreps["A1u"] = irreps["A1u"] / 2
641
642   irreps["E"] = irreps["E1"] + irreps["E2"]
643
644   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
645
646   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
647
648   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
649
650   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
651
652   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
653
654   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
655
656   irreps["A1g"] = irreps["A1g"] / 2
657
658   irreps["Eg"] = irreps["Eg"] / 2
659
660   irreps["T1g"] = irreps["T1g"] / 2
661
662   irreps["T2g"] = irreps["T2g"] / 2
663
664   irreps["T1u"] = irreps["T1u"] / 2
665
666   irreps["T2u"] = irreps["T2u"] / 2
667
668   irreps["A1u"] = irreps["A1u"] / 2
669
670   irreps["E"] = irreps["E1"] + irreps["E2"]
671
672   irreps["T_d"] = irreps["T1g"] + irreps["T2g"]
673
674   irreps["T_u"] = irreps["T1u"] + irreps["T2u"]
675
676   irreps["O_h"] = irreps["Eg"] + irreps["T1g"] + irreps["T2g"] + irreps["Eg"]
677
678   irreps["O_v"] = irreps["Eg"] + irreps["T1u"] + irreps["T2u"] + irreps["Eg"]
679
680   irreps["I_h"] = irreps["A1g"] + irreps["Eg"] + irreps["T1g"] + irreps["T2g"]
681
682   irreps["I_v"] = irreps["A1g"] + irreps["Eg"] + irreps["T1u"] + irreps["T2u"]
683
684   irreps["A1g"] = irreps["A1g"] / 2
68
```

```

32             k_point: np.ndarray, space_group: int) ->
33                 dict:
34
35             """
36             Decompose occupied states into irreducible representations.
37
38             Uses character table for little group at K.
39             """
40
41             little_group = get_little_group(k_point, space_group)
42             character_table = get_character_table(little_group)
43
44             irrep_counts = {irrep: 0 for irrep in character_table.keys()}
45
46             # For each symmetry operation g in little group
47             for g_name, g_matrix in little_group.items():
48                 # Compute character: Tr(g acting on occupied space)
49                 char_occ = np.trace(g_matrix @ states @ states.conj().T @
50                                     g_matrix.conj().T)
51
52                 # Decompose using orthogonality of characters
53                 for irrep, characters in character_table.items():
54                     irrep_counts[irrep] += char_occ *
55                         np.conj(characters[g_name])
56
57             # Normalize by group order
58             group_order = len(little_group)
59             for irrep in irrep_counts:
60                 irrep_counts[irrep] = int(round(irrep_counts[irrep].real /
61                                             group_order))
62
63             return irrep_counts

```

0.3.5 Phase 5: Robustness and Disorder (Months 6-7)

Test corner state protection:

```

1 def add_crystalline_disorder(H_func: Callable, disorder_type: str,
2                               strength: float) -> Callable:
3
4     """
5     Add disorder preserving crystalline symmetry.
6
7     disorder_type:
8     - 'C4_preserving': Disorder respects 4-fold rotation
9     - 'mirror_preserving': Respects mirror symmetries
10    - 'random': Breaks all symmetries (for comparison)
11
12    def H_disordered(k: np.ndarray) -> np.ndarray:
13        H_clean = H_func(k)
14
15        if disorder_type == 'C4_preserving':
16            # Add terms that commute with C4 operator
17            delta_H = strength * generate_c4_symmetric_perturbation()
18        elif disorder_type == 'random':
19            # Generic Hermitian perturbation
20            delta_H = strength *
21                generate_random_hermitian(H_clean.shape[0])
22        else:
23
24
25
26
27
28
29
2

```

```

21         delta_H = np.zeros_like(H_clean)
22
23     return H_clean + delta_H
24
25     return H_disordered
26
27 def test_corner_state_robustness(H_bulk: Callable, disorder_levels:
28     List[float],
29     N_trials: int = 50) -> dict:
30
31     """
32     Test corner state survival vs disorder.
33     """
34     results = {}
35
36     for disorder in disorder_levels:
37         corner_survival = []
38
39         for trial in range(N_trials):
40             H_disorder = add_crystalline_disorder(H_bulk,
41                 'C4_preserving', disorder)
42
43             corner_states, energies = find_corner_states(H_disorder)
44
45             # Check if corner states still exist
46             survival = (len(corner_states) >= 4) # 4 corners in square
47             corner_survival.append(survival)
48
49             results[disorder] = {
50                 'survival_probability': np.mean(corner_survival),
51                 'mean_corner_count': np.mean([len(find_corner_states(
52                     add_crystalline_disorder(H_bulk, 'C4_preserving',
53                     disorder))[0])
54                     for _ in range(N_trials)])}
55
56     return results

```

0.3.6 Phase 6: Classification and Database (Months 7-8)

Enumerate all HOTIs for wallpaper groups:

```

1 def classify_hotis_2d(wallpaper_group: int, max_orbitals: int = 4) ->
2     List:
3
4     """
5     Enumerate all possible 2nd-order TIs for given 2D space group.
6
7     Uses symmetry indicator theory + K-theory classification.
8     """
9     hotis = []
10
11     # Get symmetry constraints
12     point_group = get_point_group_from_space_group(wallpaper_group)
13     allowed_indicators = compute_allowed_indicators(point_group)
14
15     # Generate models for each allowed indicator
16     for indicator in allowed_indicators:

```

```

15     # Construct minimal tight-binding model realizing this indicator
16     model = construct_from_indicator(indicator, wallpaper_group,
17                                       max_orbitals)
18
19     if model is not None:
20         cert = generate_hoti_certificate(model)
21
22     if cert.quadrupole_moment != Fraction(0, 1):
23         hotis.append({
24             'space_group': wallpaper_group,
25             'indicator': indicator,
26             'quadrupole': cert.quadrupole_moment,
27             'corner_states': cert.corner_charges,
28             'model': model
29         })
30
31     return hotis
32
33 def generate_hoti_database() -> dict:
34     """
35     Generate complete database of HOTIs for all 2D wallpaper groups.
36     """
37     database = {'models': []}
38
39     # 17 wallpaper groups
40     for sg in range(1, 18):
41         print(f"Classifying space group {sg}...")
42
43         hotis = classify_hotis_2d(sg, max_orbitals=4)
44
45         for hoti in hotis:
46             cert = generate_hoti_certificate(hoti['model'])
47
48             database['models'].append({
49                 'space_group': sg,
50                 'quadrupole_moment': str(cert.quadrupole_moment),
51                 'symmetry_indicator': cert.symmetry_indicator,
52                 'corner_charge': str(cert.corner_charges[0]), # First
53                                         corner
54                 'certificate_path': export_hoti_certificate(cert)
55             })
56
57     return database

```

0.4 4. Example Starting Prompt

```

1 You are a condensed matter theorist specializing in higher-order
2 topological phases. Design
3 tight-binding models with corner/hinge states using ONLY crystalline
4 symmetryno simulations.
5
6 OBJECTIVE: Construct BBH quadrupole insulator, compute q_xy = 1/2,
7 verify corner charges e /2.

```

```

5 PHASE 1 (Months 1-2): BBH model implementation
6 - Code 4-band Hamiltonian on square lattice with C4 symmetry
7 - Verify C4 transformation:  $H(C4 \ k) = U_{C4} H(k) U_{-C4}$ 
8 - Compute bulk band structure, identify gap
9 - Test phase transition at gamma = lambda
10
11
12 PHASE 2 (Months 2-4): Nested Wilson loops
13 - Implement  $W_x(k_y) = \exp(i A_x dk_x)$ 
14 - Compute eigenphases  $\phi_i(k_y)$ 
15 - Second Wilson loop in y-direction
16 - Extract quantized polarizations  $p_x, p_y \in \{0, 1/2\}$ 
17
18 PHASE 3 (Months 4-5): Quadrupole and corners
19 - Compute  $q_{xy} = (p_x p_y - p_{-x} p_{-y})/2$ 
20 - Verify  $q_{xy} \in \{0, 1/2\}$  using exact arithmetic
21 - Solve finite 20 20 lattice for corner states
22 - Measure corner charges:  $Q_c = e/2$ 
23
24 PHASE 4 (Months 5-6): Symmetry indicators
25 - Compute irrep decomposition at (X, M, Y)
26 - Extract indicator  $z = (n_x, n_X, n_M, n_Y) \bmod 2$ 
27 - Verify formula:  $z = 0 \text{ HOTI}$ 
28
29 PHASE 5 (Months 6-7): Disorder robustness
30 - Add C4-preserving disorder:  $H \text{ with } [U_{C4}, H] = 0$ 
31 - Test corner state survival at disorder = 5%, 10%, 20%
32 - Compare to symmetry-breaking disorder
33
34 PHASE 6 (Months 7-8): Classification
35 - Enumerate HOTIs for p4 (square), p6 (hexagonal) groups
36 - Generate database with certificates
37 - Export minimal models for each topological class
38
39 SUCCESS CRITERIA:
40 - MVR: BBH model with verified  $q_{xy} = 1/2$ 
41 - Strong: Corner states computed, symmetry indicators working
42 - Publication: Complete 2D classification + database
43
44 VERIFICATION:
45 - Quadrupole moment exact:  $q_{xy} = 1/2$  (rational arithmetic)
46 - Corner charges quantized:  $Q_c = e/2$  within 1%
47 - 4 corner states for 4 corners (square geometry)
48 - Disorder threshold:  $Q_c > 15\%$  (C4-preserving)
49
50 Pure symmetry + linear algebra. No DFT, no experiments.
51 All results certificate-based with exact multipole moments.

```

0.5 5. Success Criteria

0.5.1 MVR (2-3 months)

- BBH model with $q_{xy} = 1/2$ verified

- Nested Wilson loops working

0.5.2 Strong (5-6 months)

- Corner states computed and visualized
- Symmetry indicators for 5 wallpaper groups
- Disorder robustness tested

0.5.3 Publication (7-8 months)

- Complete 2D HOTI classification
 - 3D hinge insulator examples
 - Database with certificates
-

0.6 6. Verification Protocol

Automated checks: multipole quantization, corner charge measurement, symmetry operator verification, disorder statistics.

0.7 7. Resources Milestones

References:

- Benalcazar, Bernevig, Hughes (2017): "Quantized Electric Multipole Insulators"
- Schindler et al. (2018): "Higher-Order Topological Insulators"
- Khalaf et al. (2018): "Symmetry Indicators and Anomalous Surface States"

Milestones:

- Month 2: BBH validated
 - Month 4: Nested Wilson loops extracting q_{xy}
 - Month 6: Symmetry indicators working
 - Month 8: Complete database
-

0.8 8. Extensions

- **3D Octupole Insulators**
 - **Interacting HOTIs:** Fractional corner charges
 - **Non-Hermitian HOTIs:** Exceptional points at corners
-