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 ±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:

- Designer 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):

```
q_xy = (1/2  )     _ {BZ} Tr[P (  _x   P   _y   P -   _y   P   _x   P)] dk
```

Nested Wilson Loop:

```
W_x(k_y) = exp(i   _  {0}^{2  } A_x(k_x, k_y) dk_x)
  _y (k_y) = eigenphases of W_x(k_y)
W_y = exp(i   _  {0}^{2  }   _y (k_y) dk_y)
```

Eigenphases of W$_y givequantizedpolarizationßquadrupolemoment.$

Corner Charge Formula:

```
Q_corner = e(p_x p_y - p_x - p_y) mod e
```

where px, py 0, 1/2 are bulk polarizations.

Symmetry Indicator (for space group G):

```
z = (n_  , n_X, n_M, n_Y) mod 2
```

where n$_K = (number of occupied bands with specific irrep at K) mod 2.$

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:

```python
from sympy import *
import numpy as np
from typing import List, Callable, Tuple

class CrystallineHamiltonian:
    dimension: int  # 2D or 3D
    space_group: int  # International number
    point_group: str  # Schoenflies notation (C4v, D4h, etc.)

    hamiltonian: Callable[[np.ndarray], np.ndarray]  # H(k)
    filling: int  # Number of occupied bands

    symmetry_operators: dict  # {name: unitary matrix} for C4, mirror,
        etc.
```

Output:

```python
class HOTICertificate:
    model: CrystallineHamiltonian

    # Topology
    quadrupole_moment: Fraction  # q_xy     {0, 1/2} for 2D
    octupole_moment: Optional[Fraction]  # For 3D

    nested_wilson_spectrum: List[List[float]]  #  _i ^  (k_  )
    bulk_polarizations: Tuple[Fraction, Fraction]  # (p_x, p_y)

    # Symmetry analysis
    symmetry_indicator: Tuple[int, ...]  # (n_  , n_X, ...) mod 2
    irrep_decomposition: dict  # Irrep content at each high-sym point

    # Corner/hinge states
    corner_states: List[np.ndarray]  # Wavefunctions localized to
        corners
    corner_charges: List[Fraction]  # Charge at each corner
    hinge_dispersion: Optional[np.ndarray]  # For 3D systems

    # Verification
    bulk_gap: float
    edge_gap: float  # Confirms edges are gapped
    localization_length: float  # Corner state decay into bulk

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

0.3 3. Implementation Approach

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

Implement canonical 2D quadrupole insulator:

```
import numpy as np
from sympy import *
from scipy.linalg import eigh

def bbh_model(gamma: float, lambda_param: float) -> Callable:
    """
    Benalcazar-Bernevig-Hughes (BBH) quadrupole insulator.

    2D square lattice with 4 orbitals per site.
    C4 rotation symmetry protects corner charges  e /2.

    Parameters:
    - gamma: intracell hopping (0 < gamma < 1)
    - lambda_param: intercell hopping (0 < lambda < 1)

    For gamma > lambda: trivial
    For gamma < lambda: topological (q_xy = 1/2)
    """
    def H(k: np.ndarray) -> np.ndarray:
        kx, ky = k[0], k[1]

        # Pauli matrices for sublattice
        sx = np.array([[0, 1], [1, 0]])
        sy = np.array([[0, -1j], [1j, 0]])
        sz = np.array([[1, 0], [0, -1]])
        s0 = np.eye(2)

        # Hamiltonian (4 4  = 2 orbitals    2 sublattices)
        H_k = (
            (gamma + lambda_param * np.cos(kx)) * np.kron(sx, s0) +
            (gamma + lambda_param * np.cos(ky)) * np.kron(sy, s0) +
            lambda_param * np.sin(kx) * np.kron(sz, sx) +
            lambda_param * np.sin(ky) * np.kron(sz, sy)
        )

        return H_k

    return H

def verify_c4_symmetry(H_func: Callable) -> bool:
    """
    Verify Hamiltonian has C4 rotation symmetry.

    C4: (kx, ky)     (-ky, kx)
    H(C4 k) = U_C4 H(k) U _C4
    """
    # C4 operator (90   rotation in orbital space)
    U_C4 = np.array([
        [0, 0, 0, 1],
        [1, 0, 0, 0],
        [0, 1, 0, 0],
        [0, 0, 1, 0]
    ])  # Cyclic permutation

    # Test at random k-points
    for _ in range(10):
        k = np.random.uniform(-np.pi, np.pi, 2)
        k_rot = np.array([-k[1], k[0]])  # C4 rotation in k-space

        H_k = H_func(k)
        H_k_rot = H_func(k_rot)

        # Check symmetry relation
        lhs = H_k_rot
        rhs = U_C4 @ H_k @ U_C4.conj().T

        if not np.allclose(lhs, rhs, atol=1e-10):
            return False

    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:

```
def wilson_loop_x(H_func: Callable, ky: float, band_indices: List[int],
                  N_kx: int = 100) -> np.ndarray:
    """
    Compute Wilson loop in x-direction at fixed ky.

    W_x(ky) = exp(i      A_x(kx, ky) dkx)

    Returns: Unitary matrix W_x
    """
    kx_values = np.linspace(0, 2*np.pi, N_kx, endpoint=False)
    dk_x = kx_values[1] - kx_values[0]

    # Initialize Wilson loop as identity
    W_x = np.eye(len(band_indices), dtype=complex)

    for i, kx in enumerate(kx_values):
        k = np.array([kx, ky])
        k_next = np.array([(kx + dk_x) % (2*np.pi), ky])

        # Get occupied states at k and k+dk
        evals, evecs = eigh(H_func(k))
        sorted_idx = np.argsort(evals)
        states_k = evecs[:, sorted_idx[band_indices]]

        evals_next, evecs_next = eigh(H_func(k_next))
        sorted_idx_next = np.argsort(evals_next)
        states_k_next = evecs_next[:, sorted_idx_next[band_indices]]

        # Overlap matrix
        F = states_k.conj().T @ states_k_next

        # Update Wilson loop
        W_x = W_x @ F

    return W_x

def nested_wilson_loop(H_func: Callable, band_indices: List[int],
                       N_kx: int = 100, N_ky: int = 100) -> np.ndarray:
    """
    Compute nested Wilson loop to extract quadrupole moment.

    1. Compute W_x(ky) for each ky
    2. Diagonalize to get eigenphases  _i (ky)
    3. Compute Wilson loop of  _i (ky) in ky-direction
    4. Final eigenphases give quantized polarization
    """
    ky_values = np.linspace(0, 2*np.pi, N_ky, endpoint=False)

    # Array to store eigenphases  _i (ky)
    nu_spectrum = np.zeros((N_ky, len(band_indices)))

    for j, ky in enumerate(ky_values):
        W_x = wilson_loop_x(H_func, ky, band_indices, N_kx)

        # Eigenvalues of W_x = exp(i  _i )
        eigenvalues = np.linalg.eigvals(W_x)
        nu_values = np.angle(eigenvalues)  # Phases     [-  ,   ]

        nu_spectrum[j, :] = np.sort(nu_values)

    # Now compute Wilson loop in y-direction using  _i (ky)
    # This is t r i c k y need to track which    belongs to which band

    # Simplified: compute winding of each  _i (ky)
    polarizations = []

    for i in range(len(band_indices)):
        # Winding number of  _i (ky)
        nu_traj = nu_spectrum[:, i]

        # Total phase accumulated (account for 2   jumps)
        total_phase = np.sum(np.diff(np.unwrap(nu_traj)))
        p_i = total_phase / (2*np.pi)

        polarizations.append(p_i)

    return polarizations, nu_spectrum

def compute_quadrupole_moment(H_func: Callable, band_indices:
   List[int]) -> Fraction:
```

```python
    """
    Compute quantized quadrupole moment q_xy.

    q_xy = (p_x p_y - p_x - p_y) / 2  mod 1/2

    where p_x, p_y are Wannier center polarizations.
    """
    # Compute nested Wilson loops in both directions
    p_x_list, _ = nested_wilson_loop(H_func, band_indices)
    p_y_list, _ = nested_wilson_loop(H_func, band_indices)  # Need to swap directions

    # For filled bands, take sum of polarizations mod 1
    p_x = sum(p_x_list) % 1
    p_y = sum(p_y_list) % 1

    # Quadrupole formula
    q_xy = (p_x * p_y - p_x - p_y) / 2

    # Quantize to {0, 1/2}
    if abs(q_xy) < 0.25:
        return Fraction(0, 1)
    elif abs(q_xy - 0.5) < 0.25 or abs(q_xy + 0.5) < 0.25:
        return Fraction(1, 2)
    else:
        # Should not happen for topological systems
        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:

```python
def finite_lattice_hamiltonian(H_bulk: Callable, L_x: int, L_y: int) -> np.ndarray:
    """
    Construct Hamiltonian for finite L_x    L_y lattice with open boundaries.

    Each unit cell has N_orb orbitals.
    Total Hilbert space dimension: N_orb    L_x    L_y
    """
    # Get unit cell Hamiltonian dimension
    H_test = H_bulk(np.array([0, 0]))
    N_orb = H_test.shape[0]

    dim = N_orb * L_x * L_y
    H_finite = np.zeros((dim, dim), dtype=complex)

    for ix in range(L_x):
        for iy in range(L_y):
            # On-site terms
            idx = (ix * L_y + iy) * N_orb

            # Intracell Hamiltonian (k=0 term)
            H_00 = H_bulk(np.array([0, 0]))
            H_finite[idx:idx+N_orb, idx:idx+N_orb] = H_00

            # Hopping in x-direction
            if ix < L_x - 1:
                idx_next_x = ((ix+1) * L_y + iy) * N_orb

                # Extract hopping from k-dependence
                H_kx = H_bulk(np.array([np.pi/L_x, 0]))  # Small kx
                t_x = (H_kx - H_00) / (1j * np.pi/L_x)  # Linear term

                H_finite[idx:idx+N_orb, idx_next_x:idx_next_x+N_orb] =
                   t_x
                H_finite[idx_next_x:idx_next_x+N_orb, idx:idx+N_orb] =
                   t_x.conj().T

            # Hopping in y-direction
            if iy < L_y - 1:
                idx_next_y = (ix * L_y + (iy+1)) * N_orb

                H_ky = H_bulk(np.array([0, np.pi/L_y]))
                t_y = (H_ky - H_00) / (1j * np.pi/L_y)

                H_finite[idx:idx+N_orb, idx_next_y:idx_next_y+N_orb] =
                   t_y
                H_finite[idx_next_y:idx_next_y+N_orb, idx:idx+N_orb] =
                   t_y.conj().T

    return H_finite

def find_corner_states(H_bulk: Callable, L_x: int = 20, L_y: int = 20,
                       energy_threshold: float = 0.01) ->
                          List[np.ndarray]:
    """
    Find in-gap corner states for finite system.
    """
    H_finite = finite_lattice_hamiltonian(H_bulk, L_x, L_y)

    # Diagonalize
    eigenvalues, eigenvectors = eigh(H_finite)

    # Find states near zero energy (in gap)
    gap_indices = np.where(np.abs(eigenvalues) < energy_threshold)[0]

    corner_states = [eigenvectors[:, idx] for idx in gap_indices]

    return corner_states, eigenvalues[gap_indices]

def compute_corner_charge(corner_state: np.ndarray, L_x: int, L_y: int,
                          N_orb: int) -> Fraction:
    """
    Compute charge localized at corner.

    Integrate |  |   in corner region (e.g., 5 5  sites around corner).
    """
    # Reshape wavefunction to lattice
    psi_lattice = corner_state.reshape((L_x, L_y, N_orb))

```

```python
    # Define corner region (bottom-left as example)
    corner_size = min(5, L_x//4, L_y//4)

    corner_charge = 0
    for ix in range(corner_size):
        for iy in range(corner_size):
            # Sum over orbitals
            corner_charge += np.sum(np.abs(psi_lattice[ix, iy, :])**2)

    # Quantize (should be      1/2 for HOTI)
    if abs(corner_charge - 0.5) < 0.1:
        return Fraction(1, 2)
    elif abs(corner_charge) < 0.1:
        return Fraction(0, 1)
    else:
        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:

```python
def compute_symmetry_indicator(H_func: Callable, space_group: int,
                               band_indices: List[int]) -> Tuple[int, ...]:
    """
    Compute symmetry indicator z = (n_  , n_X, n_M, n_Y) mod 2.

    For each high-symmetry point K, count occupied bands with specific irreps.
    """
    # Get high-symmetry points for space group
    high_sym_points = get_high_symmetry_points_2d(space_group)

    indicators = []

    for K_name, k_point in high_sym_points:
        H_K = H_func(k_point)
        evals, evecs = eigh(H_K)

        # Get occupied states
        sorted_idx = np.argsort(evals)
        occupied_states = evecs[:, sorted_idx[band_indices]]

        # Determine irrep content using character table
        irrep_counts = decompose_into_irreps(H_K, occupied_states, k_point, space_group)

        # Specific indicator: e.g., number of A1g reps mod 2
        n_K = irrep_counts['A1g'] % 2  # Convention depends on space group

        indicators.append(n_K)

    return tuple(indicators)

def decompose_into_irreps(H_K: np.ndarray, states: np.ndarray,
                              k_point: np.ndarray, space_group: int) ->
                                  dict:
    """
    Decompose occupied states into irreducible representations.

    Uses character table for little group at K.
    """
    little_group = get_little_group(k_point, space_group)
    character_table = get_character_table(little_group)

    irrep_counts = {irrep: 0 for irrep in character_table.keys()}

    # For each symmetry operation g in little group
    for g_name, g_matrix in little_group.items():
        # Compute character: Tr(g acting on occupied space)
        char_occ = np.trace(g_matrix @ states @ states.conj().T @
            g_matrix.conj().T)

        # Decompose using orthogonality of characters
        for irrep, characters in character_table.items():
            irrep_counts[irrep] += char_occ *
                np.conj(characters[g_name])

    # Normalize by group order
    group_order = len(little_group)
    for irrep in irrep_counts:
        irrep_counts[irrep] = int(round(irrep_counts[irrep].real /
            group_order))

    return irrep_counts
```

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

Test corner state protection:

```python
def add_crystalline_disorder(H_func: Callable, disorder_type: str,
                              strength: float) -> Callable:
    """
    Add disorder preserving crystalline symmetry.

    disorder_type:
    - 'C4_preserving': Disorder respects 4-fold rotation
    - 'mirror_preserving': Respects mirror symmetries
    - 'random': Breaks all symmetries (for comparison)
    """
    def H_disordered(k: np.ndarray) -> np.ndarray:
        H_clean = H_func(k)

        if disorder_type == 'C4_preserving':
            # Add terms that commute with C4 operator
            delta_H = strength * generate_c4_symmetric_perturbation()
        elif disorder_type == 'random':
            # Generic Hermitian perturbation
            delta_H = strength *
                generate_random_hermitian(H_clean.shape[0])
        else:
            delta_H = np.zeros_like(H_clean)

        return H_clean + delta_H

    return H_disordered

def test_corner_state_robustness(H_bulk: Callable, disorder_levels: List[float],
                                 N_trials: int = 50) -> dict:
    """
    Test corner state survival vs disorder.
    """
    results = {}

    for disorder in disorder_levels:
        corner_survival = []

        for trial in range(N_trials):
            H_disorder = add_crystalline_disorder(H_bulk, 'C4_preserving', disorder)

            corner_states, energies = find_corner_states(H_disorder)

            # Check if corner states still exist
            survival = (len(corner_states) >= 4)  # 4 corners in square
            corner_survival.append(survival)

        results[disorder] = {
            'survival_probability': np.mean(corner_survival),
            'mean_corner_count': np.mean([len(find_corner_states(
                add_crystalline_disorder(H_bulk, 'C4_preserving', disorder))[0])
                for _ in range(N_trials)])
        }

    return results
```

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

Enumerate all HOTIs for wallpaper groups:

```
def classify_hotis_2d(wallpaper_group: int, max_orbitals: int = 4) -> List:
    """
    Enumerate all possible 2nd-order TIs for given 2D space group.

    Uses symmetry indicator theory + K-theory classification.
    """
    hotis = []

    # Get symmetry constraints
    point_group = get_point_group_from_space_group(wallpaper_group)
    allowed_indicators = compute_allowed_indicators(point_group)

    # Generate models for each allowed indicator
    for indicator in allowed_indicators:
```

```python
        # Construct minimal tight-binding model realizing this indicator
        model = construct_from_indicator(indicator, wallpaper_group, max_orbitals)

        if model is not None:
            cert = generate_hoti_certificate(model)

            if cert.quadrupole_moment != Fraction(0, 1):
                hotis.append({
                    'space_group': wallpaper_group,
                    'indicator': indicator,
                    'quadrupole': cert.quadrupole_moment,
                    'corner_states': cert.corner_charges,
                    'model': model
                })

    return hotis

def generate_hoti_database() -> dict:
    """
    Generate complete database of HOTIs for all 2D wallpaper groups.
    """
    database = {'models': []}

    # 17 wallpaper groups
    for sg in range(1, 18):
        print(f"Classifying space group {sg}...")

        hotis = classify_hotis_2d(sg, max_orbitals=4)

        for hoti in hotis:
            cert = generate_hoti_certificate(hoti['model'])

            database['models'].append({
                'space_group': sg,
                'quadrupole_moment': str(cert.quadrupole_moment),
                'symmetry_indicator': cert.symmetry_indicator,
                'corner_charge': str(cert.corner_charges[0]),  # First corner
                'certificate_path': export_hoti_certificate(cert)
            })

    return database
```

0.4 4. Example Starting Prompt

```
You are a condensed matter theorist specializing in higher-order topological phases. Design
tight-binding models with corner/hinge states using ONLY crystalline symmetryno simulations.

OBJECTIVE: Construct BBH quadrupole insulator, compute q_xy = 1/2, verify corner charges  e /2.
```

```

PHASE 1 (Months 1-2): BBH model implementation
- Code 4-band Hamiltonian on square lattice with C4 symmetry
- Verify C4 transformation: H(C4 k) = U_C4 H(k) U_C4
- Compute bulk band structure, identify gap
- Test phase transition at gamma = lambda

PHASE 2 (Months 2-4): Nested Wilson loops
- Implement W_x(ky) = exp(i     A_x dk_x)
- Compute eigenphases  _i (ky)
- Second Wilson loop in y-direction
- Extract quantized polarizations p_x, p_y     {0, 1/2}

PHASE 3 (Months 4-5): Quadrupole and corners
- Compute q_xy = (p_x p_y - p_x - p_y)/2
- Verify q_xy     {0, 1/2} using exact arithmetic
- Solve finite 20 20  lattice for corner states
- Measure corner charges: Q_c      e /2

PHASE 4 (Months 5-6): Symmetry indicators
- Compute irrep decomposition at (  , X, M, Y)
- Extract indicator z = (n_  , n_X, n_M, n_Y) mod 2
- Verify formula: z     0     HOTI

PHASE 5 (Months 6-7): Disorder robustness
- Add C4-preserving disorder:  H  with [U_C4,  H ] = 0
- Test corner state survival at    = 5%, 10%, 20%
- Compare to symmetry-breaking disorder

PHASE 6 (Months 7-8): Classification
- Enumerate HOTIs for p4 (square), p6 (hexagonal) groups
- Generate database with certificates
- Export minimal models for each topological class

SUCCESS CRITERIA:
- MVR: BBH model with verified q_xy = 1/2
- Strong: Corner states computed, symmetry indicators working
- Publication: Complete 2D classification + database

VERIFICATION:
- Quadrupole moment exact: q_xy = 1/2 (rational arithmetic)
- Corner charges quantized: Q_c =  e /2 within 1%
- 4 corner states for 4 corners (square geometry)
- Disorder threshold:  _c  > 15% (C4-preserving)

Pure symmetry + linear algebra. No DFT, no experiments.
All results certificate-based with exact multipole moments.
```

0.5 5. Success Criteria

0.5.1 MVR (2-3 months)

- BBH model with q$_x y = 1/2 verified$

- 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$_x y$
- 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

End of PRD 13