PRD 09: Topological Band Theory Without Materials Data

Pure Thought AI Challenge 09

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: 3-6 months
Difficulty: Medium-High
Prerequisites: Solid state physics, algebraic topology, group theory, computational linear algebra

---

0.1 1. Problem Statement

0.1.1 Scientific Context

Topological band theory has revolutionized condensed matter physics by identifying materials where electronic properties are protected by topology rather than symmetry. These topological invariants—such as Chern numbers, indices, and winding numbers—are robust to disorder and perturbations, making them ideal for quantum technologies.

Traditional materials discovery relies heavily on:

- **Databases**: Materials Project, ICSD, AFLOW containing 100,000+ known crystal structures
- **DFT Calculations**: Expensive ab initio computations to determine band structures
- **Experimental Validation**: Synthesis and measurement to confirm predictions

However, **pure-thought approaches** can:

- Generate topological band structures from symmetry principles alone
- Prove existence of specific topological phases without materials realization
- Discover minimal tight-binding models exhibiting non-trivial topology
- Classify all possible topological phases for a given space group

Key topological invariants:

- **Chern Number** (C): Integer classifying 2D insulators (quantum Hall effect)
- **Index** (): Binary invariant for time-reversal invariant systems (topological insulators)
- **Winding Number** (W): 1D invariant for SSH models
- **Monopole Charge** (Q): 3D Weyl semimetals

0.1.2 Core Question

Can we systematically discover tight-binding models with non-trivial topological invariants using only symmetry constraints, algebraic topology, and exact computation—without reference to any experimental materials data?

Specifically:

- Given a space group G and filling fraction, enumerate all tight-binding Hamiltonians with non-zero Chern number
- Construct minimal models (fewest orbitals/sites) for each topological class
- Certify topological invariants using exact arithmetic and algebraic geometry
- Prove lower bounds on the number of bands required for specific topology

0.1.3 Why This Matters

Theoretical Impact:

- Establishes pure mathematics as a predictive tool for materials science
- Reveals universal patterns in topological phase diagrams
- Identifies "inevitable" topological phases that must exist in nature

Practical Benefits:

- Guides experimental synthesis toward high-probability targets
- Discovers models for quantum simulation on cold atoms/photonic lattices
- Provides blueprints for topological quantum computing platforms

Pure Thought Advantages:

- Brillouin zone integrals can be computed exactly using algebraic topology
- Symmetry analysis is purely group-theoretic
- K-theory classifications are rigorous mathematical theorems
- No expensive DFT calculations or experimental databases needed

---

0.2 2. Mathematical Formulation

0.2.1 Problem Definition

A **tight-binding Hamiltonian** on a lattice with Bravais vectors a, a, ..., a is:

```
H(k) =  _ {  ,  }  _ {R} t_{    }(R) exp(i  k  R ) |          |
```

where:

- k BZ is the crystal momentum (Brillouin zone)
- , label orbitals (s, p, d, ...) at each site
- R are lattice vectors
- t(R)$arehoppingamplitudes$

For an insulator with filled bands n, n, ..., n, the **Chern number** is:

```
C = (1/2 i )   _  {BZ} Tr[P (     P           P      -      P
        P     )] dk_x dk_y
```

where P(k) = |u(k)⟩⟨u(k)| is the projector onto occupied bands.
Certificate of Non-Trivial Topology:
Given H(k), we provide:

- **Exact Chern Number**: Computed via k-space discretization and Berry curvature integration
- **Edge State Spectrum**: Demonstration of protected boundary modes
- **Symmetry Analysis**: Proof that C 0 is allowed by space group
- **Minimal Model**: Proof that fewer orbitals/sites cannot achieve this C

0.2.2 Constraints

- **Hermiticity**: H(k)† = H(k)
- **Time-Reversal** (if applicable): T H(k) T^{1} = H(-k) with $T^{2} = \pm 1$
- **Inversion** (if applicable): I H(k) I^{1} = H(-k)
- **Space Group Symmetry**: g H(k) g^{1} = H(g · k) for g G
- **Band Gap**: Energy gap > 0 between occupied and unoccupied bands
- **Filling Constraint**: Exactly N bands filled (N < total number of bands)

0.2.3 Input/Output Specification

Input:

```python
from sympy import Symbol, Matrix, exp, I, pi, cos, sin
from typing import List, Tuple, Callable

class LatticeSymmetry:
    space_group: int  # International space group number (1-230)
    point_group: str  # Schoenflies notation (C4v, D6h, etc.)
    dimension: int    # 1D, 2D, or 3D
    bravais_vectors: List[np.ndarray]

class TightBindingModel:
    num_orbitals: int  # Orbitals per unit cell
    hamiltonian: Callable[[np.ndarray], np.ndarray]  # H(k) as function
    symmetry: LatticeSymmetry
    filling: int  # Number of filled bands
```

Output:

```python
class TopologicalCertificate:
    model: TightBindingModel
    chern_number: int  # Exact integer
    z2_index: Optional[int]  # For time-reversal invariant systems

    # Verification data
    berry_curvature: Callable[[np.ndarray], float]  # F(k) =      A
       -      A
    berry_connection: Callable[[np.ndarray], np.ndarray]  # A(k)

    edge_spectrum: np.ndarray  # Edge state energies
    edge_localization: np.ndarray  # Wavefunction decay length

    symmetry_proof: str  # Proof that topology is symmetry-allowed
    minimality_proof: str  # Proof of minimal orbital count

    # Exact computation artifacts
    discretization_error: float  # Should be < 1e-10
    gap_minimum: float  # min_{k} [E_{N+1}(k) - E_N(k)]
```

0.3 3. Implementation Approach

0.3.1 Phase 1: Symmetry-Constrained Model Generation (Month 1)

Build infrastructure for generating tight-binding models respecting space group symmetries:

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

def square_lattice_hamiltonian(t1: float, t2: float, phi: float) ->
    Callable:
    """
    Haldane-like model on square lattice.

    t1: Nearest-neighbor hopping
    t2: Next-nearest-neighbor hopping
    phi: Magnetic flux through plaquette (breaks time-reversal)

    Returns H(kx, ky) as 2 2  matrix (two sublattices).
    """
    def H(k: np.ndarray) -> np.ndarray:
        kx, ky = k[0], k[1]

        # Nearest-neighbor terms
        h_nn = 2*t1 * (np.cos(kx) + np.cos(ky))

        # Next-nearest-neighbor with complex phase
        h_nnn = 2*t2 * (np.cos(kx + ky + phi) + np.cos(kx - ky + phi))

        # Off-diagonal coupling
        h_od = t1 * (np.exp(1j*kx) + np.exp(1j*ky))

        return np.array([
            [h_nn + h_nnn, h_od],
            [np.conj(h_od), h_nn + h_nnn]
        ], dtype=complex)

    return H

def honeycomb_haldane_model(t: float, t2: float, phi: float, M: float)
    -> Callable:
    """
    Original Haldane model on honeycomb lattice.

    t: Nearest-neighbor hopping
    t2: Next-nearest-neighbor hopping
    phi: Magnetic flux (preserves time-reversal on average)
    M: Staggered sublattice potential

    Exhibits Chern insulator phase for M < 3  3   t2 sin(phi).
    """
    def H(k: np.ndarray) -> np.ndarray:
        kx, ky = k[0], k[1]

        # Nearest-neighbor phase
```

```
        f_k = t * (1 + np.exp(1j*kx) + np.exp(1j*ky))

        # Next-nearest-neighbor phase
        chi_k = 2*t2*np.sin(phi) * (
            np.sin(kx) + np.sin(ky) - np.sin(kx + ky)
        )

        return np.array([
            [M + chi_k, f_k],
            [np.conj(f_k), -M - chi_k]
        ], dtype=complex)

    return H

def generate_symmetric_hamiltonians(symmetry: LatticeSymmetry,
                                    num_orbitals: int,
                                    max_neighbors: int = 3) ->
                                       List[TightBindingModel]:
    """
    Enumerate all tight-binding Hamiltonians respecting given space
       group.

    Uses representation theory to determine allowed hopping terms.
    """
    models = []

    # Get irreducible representations of space group
    irreps = get_space_group_irreps(symmetry.space_group)

    # Enumerate hopping terms up to max_neighbors
    for neighbor_order in range(1, max_neighbors + 1):
        # Get symmetry-allowed hopping vectors
        hopping_vectors = get_symmetric_hoppings(
            symmetry.bravais_vectors,
            symmetry.point_group,
            neighbor_order
        )

        # Parameterize Hamiltonian with symbolic amplitudes
        params = symbols(f't1:{len(hopping_vectors)+1}', real=True)

        def H_symbolic(k, params_val):
            H_mat = zeros(num_orbitals, num_orbitals)
            for i, (R, t) in enumerate(zip(hopping_vectors,
                params_val)):
                phase = exp(I * (k[0]*R[0] + k[1]*R[1]))
                # Add hopping term (orbital structure depends on irreps)
                H_mat += t * phase * orbital_matrix(i, irreps)
            return H_mat + H_mat.H  # Ensure Hermiticity

        models.append(TightBindingModel(
            num_orbitals=num_orbitals,
            hamiltonian=lambda k: H_symbolic(k, params),
            symmetry=symmetry,
            filling=num_orbitals // 2
        ))
```

```
    return models
```

Validation: Reproduce Haldane's honeycomb model, verify C = ±1 for specific parameters.

0.3.2 Phase 2: Berry Curvature and Chern Number Calculation (Months 1-2)

Implement exact computation of topological invariants:

```
def berry_connection(H_func: Callable, k: np.ndarray,
                     band_indices: List[int], delta: float = 1e-6) ->
                        np.ndarray:
    """
    Compute Berry connection A_  (k) = i   u_n  (k)|    _    |u_n(k)    .

    Uses finite differences for derivatives.
    """
    dim = len(k)
    evals, evecs = eigh(H_func(k))

    # Sort by energy
    sorted_idx = np.argsort(evals)
    occupied_states = evecs[:, sorted_idx[band_indices]]

    A = np.zeros(dim, dtype=complex)

    for mu in range(dim):
        dk = np.zeros(dim)
        dk[mu] = delta

        # Forward difference
        evals_plus, evecs_plus = eigh(H_func(k + dk))
        sorted_idx_plus = np.argsort(evals_plus)
        occupied_plus = evecs_plus[:, sorted_idx_plus[band_indices]]

        # Berry connection via overlap
        overlap = occupied_states.conj().T @ occupied_plus
        A[mu] = 1j * np.log(np.linalg.det(overlap)) / delta

    return A

def berry_curvature_2d(H_func: Callable, k: np.ndarray,
                       band_indices: List[int]) -> float:
    """
    Compute Berry curvature F_{xy}(k) =   _x   A_y -   _y   A_x.

    Returns exact value using gauge-invariant formula.
    """
    delta = 1e-6

    # Compute A at four nearby points (plaquette)
    A_00 = berry_connection(H_func, k, band_indices, delta)
    A_10 = berry_connection(H_func, k + [delta, 0], band_indices, delta)
    A_01 = berry_connection(H_func, k + [0, delta], band_indices, delta)
    A_11 = berry_connection(H_func, k + [delta, delta], band_indices,
        delta)
```

```
46
47      # Lattice curl
48      F_xy = (A_10[1] - A_00[1]) / delta - (A_01[0] - A_00[0]) / delta
49
50      return F_xy.real
51
52  def compute_chern_number(H_func: Callable, band_indices: List[int],
53                           N_k: int = 100) -> Tuple[int, float]:
54      """
55      Compute Chern number C via Brillouin zone integration.
56
57      C = (1/2  )   _  {BZ} F_{xy}(k)  d  k
58
59      Returns (C_rounded, discretization_error).
60      """
61      # Discretize Brillouin zone
62      kx_grid = np.linspace(-np.pi, np.pi, N_k, endpoint=False)
63      ky_grid = np.linspace(-np.pi, np.pi, N_k, endpoint=False)
64
65      C_integral = 0.0
66
67      for kx in kx_grid:
68          for ky in ky_grid:
69              k = np.array([kx, ky])
70              F = berry_curvature_2d(H_func, k, band_indices)
71              C_integral += F
72
73      # Normalize
74      C_integral *= (2*np.pi / N_k)**2 / (2*np.pi)
75
76      C_rounded = int(np.round(C_integral))
77      error = abs(C_integral - C_rounded)
78
79      return C_rounded, error
80
81  def compute_chern_number_exact(H_func: Callable, band_indices:
        List[int]) -> int:
82      """
83      Compute Chern number using Fukui-Hatsugai-Suzuki method (lattice
            gauge theory).
84
85      Gives exact integer result without discretization error.
86      """
87      N_k = 200  # Fine grid
88
89      # Discretize BZ on square lattice
90      kx_grid = np.linspace(-np.pi, np.pi, N_k, endpoint=False)
91      ky_grid = np.linspace(-np.pi, np.pi, N_k, endpoint=False)
92
93      # Compute link variables U_  (k) on lattice
94      U_total = 1.0 + 0j
95
96      for i, kx in enumerate(kx_grid):
97          for j, ky in enumerate(ky_grid):
98              k = np.array([kx, ky])
99
```

```
            # Get occupied states at four corners of plaquette
            states_00 = get_occupied_states(H_func, k, band_indices)
            states_10 = get_occupied_states(H_func, k + [2*np.pi/N_k,
               0], band_indices)
            states_01 = get_occupied_states(H_func, k + [0,
               2*np.pi/N_k], band_indices)
            states_11 = get_occupied_states(H_func, k + [2*np.pi/N_k,
               2*np.pi/N_k], band_indices)

            # Link variables (overlap matrices)
            U_x = np.linalg.det(states_00.conj().T @ states_10)
            U_y = np.linalg.det(states_10.conj().T @ states_11)
            U_x_inv = np.linalg.det(states_11.conj().T @ states_01)
            U_y_inv = np.linalg.det(states_01.conj().T @ states_00)

            # Plaquette product
            U_plaquette = U_x * U_y * U_x_inv * U_y_inv
            U_total *= U_plaquette

    # Chern number from total phase
    C = int(np.round(np.angle(U_total) / (2*np.pi)))

    return C
```

Validation:

- Haldane model: C = 1 for M/t2 < 33 sin()
- Qi-Wu-Zhang model: C = ±1 depending on parameters
- Square lattice with flux: C = number of flux quanta

0.3.3 Phase 3: Edge State Calculation (Months 2-3)

Compute topologically protected edge modes:

```
def ribbon_hamiltonian(H_bulk: Callable, width: int,
                       edge_direction: str = 'x') -> Callable:
    """
    Construct ribbon geometry with open boundary in one direction.

    width: Number of unit cells in finite direction
    edge_direction: 'x' or 'y' for which direction to cut
    """
    def H_ribbon(k_parallel: float) -> np.ndarray:
        """
        H_ribbon(k) has dimension (width    num_orbitals)    (width
           num_orbitals)
        """
        num_orb = H_bulk(np.array([0, 0])).shape[0]
        dim = width * num_orb
        H_rib = np.zeros((dim, dim), dtype=complex)

        for i in range(width):
            for j in range(width):
                if edge_direction == 'x':
```

```
                    # k_parallel is ky, finite direction is x
                    k_eff = np.array([0, k_parallel])
                else:
                    # k_parallel is kx, finite direction is y
                    k_eff = np.array([k_parallel, 0])

                # Hopping within layer
                if i == j:
                    H_rib[i*num_orb:(i+1)*num_orb,
                          j*num_orb:(j+1)*num_orb] = \
                           H_bulk(k_eff)

                # Hopping between layers
                elif abs(i - j) == 1:
                    t_perp = get_interlayer_hopping(H_bulk,
                        edge_direction)
                    H_rib[i*num_orb:(i+1)*num_orb,
                          j*num_orb:(j+1)*num_orb] = t_perp

        return H_rib

    return H_ribbon

def compute_edge_spectrum(H_bulk: Callable, width: int = 50,
                          N_k: int = 200) -> np.ndarray:
    """
    Compute energy spectrum of ribbon geometry.

    Returns array of shape (N_k, width * num_orbitals) with all
       eigenvalues.
    """
    H_rib = ribbon_hamiltonian(H_bulk, width)

    k_parallel = np.linspace(-np.pi, np.pi, N_k)
    spectrum = np.zeros((N_k, width * H_bulk(np.array([0,0])).shape[0]))

    for i, k in enumerate(k_parallel):
        evals = np.linalg.eigvalsh(H_rib(k))
        spectrum[i, :] = evals

    return spectrum

def identify_edge_states(spectrum: np.ndarray, gap_threshold: float =
    0.1) -> np.ndarray:
    """
    Identify edge states as those with energies in the bulk gap.
    """
    # Find bulk gap
    E_min = np.min(spectrum)
    E_max = np.max(spectrum)

    # Histogram to find gap
    hist, bins = np.histogram(spectrum.flatten(), bins=1000)
    gap_center = bins[np.argmin(hist)]

    # Extract edge states
```

```
    edge_mask = np.abs(spectrum - gap_center) < gap_threshold
    edge_energies = spectrum[edge_mask]

    return edge_energies
```

Validation:

- Verify edge states appear for C 0 models
- Check edge state chirality (left-moving vs right-moving)
- Confirm edge state count matches bulk Chern number

0.3.4 Phase 4: Systematic Space Group Classification (Months 3-4)

Enumerate topological phases for each 2D space group:

```
def classify_topological_phases(space_group: int,
                                num_orbitals: int = 2,
                                max_chern: int = 3) ->
                                   List[TopologicalCertificate]:
    """
    Enumerate all tight-binding models with |C|     max_chern for given
       space group.
    """
    symmetry = LatticeSymmetry(
        space_group=space_group,
        point_group=get_point_group(space_group),
        dimension=2,
        bravais_vectors=get_bravais_vectors(space_group)
    )

    # Generate symmetry-allowed models
    candidate_models = generate_symmetric_hamiltonians(
        symmetry, num_orbitals, max_neighbors=3
    )

    certificates = []

    for model in candidate_models:
        # Scan parameter space
        param_grid = np.linspace(-1, 1, 20)  # For each hopping
           parameter

        for params in itertools.product(param_grid,
           repeat=len(model.parameters)):
            H_func = lambda k: model.hamiltonian(k, params)

            # Check if system has a gap
            gap = compute_band_gap(H_func, model.filling)
            if gap < 0.1:
                continue

            # Compute Chern number
            band_indices = list(range(model.filling))
            C = compute_chern_number_exact(H_func, band_indices)
```

```
36
37              if abs(C) <= max_chern and C != 0:
38                  # Found non-trivial topology!
39                  cert = TopologicalCertificate(
40                      model=model,
41                      chern_number=C,
42                      berry_curvature=lambda k:
                            berry_curvature_2d(H_func, k, band_indices),
43                      edge_spectrum=compute_edge_spectrum(H_func),
44                      symmetry_proof=prove_topology_allowed(symmetry, C),
45                      gap_minimum=gap
46                  )
47                  certificates.append(cert)
48
49      return certificates
50
51  def prove_topology_allowed(symmetry: LatticeSymmetry, C: int) -> str:
52      """
53      Prove that Chern number C is compatible with space group symmetry.
54
55      Uses compatibility relations between k-space symmetries and
            topology.
56      """
57      proof = f"Space Group {symmetry.space_group} Analysis:\n"
58
59      # Check time-reversal
60      if has_time_reversal(symmetry.point_group):
61          if C != 0:
62              return "FORBIDDEN: Time-reversal symmetry requires C = 0"
63
64      # Check inversion symmetry
65      if has_inversion(symmetry.point_group):
66          # Inversion allows non-zero C
67          proof += f"    Inversion symmetry present, C = {C} allowed\n"
68
69      # Check rotation symmetries
70      rotation_order = get_max_rotation_order(symmetry.point_group)
71      if rotation_order > 1:
72          proof += f"    C_{rotation_order} rotation symmetry, C mod
                {rotation_order} = {C % rotation_order}\n"
73
74      # Check compatibility with high-symmetry points
75      high_sym_points = get_high_symmetry_points(symmetry.space_group)
76      for point_name, k_point in high_sym_points:
77          little_group = get_little_group(k_point, symmetry.space_group)
78          proof += f"    {point_name}: little group {little_group}\n"
79
80      return proof
```

Deliverable: Database of all topological phases for space groups p1, p2, p4, p6 (2D wallpaper groups).

0.3.5 Phase 5: Minimal Model Discovery (Months 4-5)

Find the smallest tight-binding models for each topological class:

```python
def find_minimal_chern_model(target_C: int) -> Tuple[TightBindingModel, str]:
    """
    Find tight-binding model with Chern number C using fewest orbitals/sites.

    Returns (model, proof_of_minimality).
    """
    # Known lower bounds from obstruction theory
    min_orbitals_required = {
        1: 2,   # C=1 requires at least 2 bands (Haldane model)
        2: 3,   # C=2 requires at least 3 bands
        3: 4,   # C=3 requires at least 4 bands (general pattern: |C| ≤ N-1)
    }

    min_orb = min_orbitals_required.get(abs(target_C), abs(target_C) + 1)

    # Search starting from minimal orbital count
    for num_orbitals in range(min_orb, 10):
        for lattice_type in ['square', 'honeycomb', 'triangular']:
            models = generate_all_models(lattice_type, num_orbitals)

            for model in models:
                C = compute_chern_number_exact(model.hamiltonian,
                                               list(range(num_orbitals // 2)))

                if C == target_C:
                    # Found minimal model!
                    proof = prove_minimality(model, target_C)
                    return model, proof

    raise ValueError(f"No model found for C = {target_C}")

def prove_minimality(model: TightBindingModel, C: int) -> str:
    """
    Prove that fewer orbitals cannot achieve Chern number C.

    Uses K-theory obstructions and index theorems.
    """
    N = model.num_orbitals

    proof = f"Minimality Proof for C = {C} with {N} orbitals:\n\n"

    # Obstruction 1: Chern number bounded by number of bands
    proof += f"1. General bound: |C| ≤ N - 1 for N-band model\n"
    proof += f"   Required: {abs(C)} ≤ {N - 1}    \n\n"

    # Obstruction 2: Parity constraints
    if C % 2 == 1:
        proof += f"2. Odd Chern number requires breaking time-reversal symmetry\n"
        proof += f"   Model breaks TRS: {not has_time_reversal(model.symmetry.point_group)}    \n\n"
```

```
    # Obstruction 3: Monopole charge in k-space
    proof += f"3. K-theory classification: [H(k)]      K  (T  )
          \n"
    proof += f"   Chern number is complete topological invariant
          \n\n"

    # Obstruction 4: Explicit check that N-1 orbitals fail
    if N > 2:
        proof += f"4. Explicit check with {N-1} orbitals:\n"
        smaller_models = generate_all_models(model.symmetry.dimension,
            N - 1)
        max_C_smaller = max([compute_chern_number_exact(m.hamiltonian,
            list(range((N-1)//2)))
                             for m in smaller_models])
        proof += f"   Maximum |C| achievable: {max_C_smaller} <
            {abs(C)}    \n\n"

    proof += f"Conclusion: {N} orbitals is minimal for C = {C}."

    return proof
```

0.3.6 Phase 6: Export and Verification (Months 5-6)

Generate comprehensive database with certificates:

```
def generate_topological_database(max_chern: int = 5) -> dict:
    """
    Generate complete database of topological tight-binding models.
    """
    database = {
        'timestamp': datetime.now().isoformat(),
        'models': [],
        'statistics': {}
    }

    chern_counts = {}

    for C in range(-max_chern, max_chern + 1):
        if C == 0:
            continue

        print(f"Finding minimal model for C = {C}...")
        model, proof = find_minimal_chern_model(C)

        cert = TopologicalCertificate(
            model=model,
            chern_number=C,
            berry_curvature=lambda k:
                berry_curvature_2d(model.hamiltonian, k,
                                   list(range(model.filling))),
            edge_spectrum=compute_edge_spectrum(model.hamiltonian),
            symmetry_proof=prove_topology_allowed(model.symmetry, C),
            minimality_proof=proof,
            gap_minimum=compute_band_gap(model.hamiltonian,
                model.filling)
```

```
        )

        database['models'].append({
            'chern_number': C,
            'num_orbitals': model.num_orbitals,
            'lattice_type': model.symmetry.bravais_vectors,
            'space_group': model.symmetry.space_group,
            'gap': cert.gap_minimum,
            'edge_state_count': count_edge_states(cert.edge_spectrum),
            'certificate_path': export_certificate(cert,
               f'chern_{C}.json')
        })

        chern_counts[C] = chern_counts.get(C, 0) + 1

    database['statistics'] = {
        'total_models': len(database['models']),
        'chern_distribution': chern_counts,
        'max_gap': max([m['gap'] for m in database['models']]),
        'min_orbitals': min([m['num_orbitals'] for m in
           database['models']])
    }

    return database

def export_certificate(cert: TopologicalCertificate, filename: str):
    """Export certificate with all verification data."""
    import json

    cert_dict = {
        'chern_number': int(cert.chern_number),
        'num_orbitals': cert.model.num_orbitals,
        'gap_minimum': float(cert.gap_minimum),
        'symmetry_proof': cert.symmetry_proof,
        'minimality_proof': cert.minimality_proof,
        'edge_state_energies': cert.edge_spectrum.tolist(),
        'discretization_error': float(cert.discretization_error)
    }

    with open(filename, 'w') as f:
        json.dump(cert_dict, f, indent=2)

    return filename
```

0.4 4. Example Starting Prompt

```
You are a computational condensed matter physicist specializing in
   topological band theory.
Your task is to discover tight-binding models with non-trivial topology
   using ONLY symmetry
principles and exact computationno experimental materials databases
   allowed.

```

```
OBJECTIVE: Systematically construct tight-binding Hamiltonians with Chern numbers C = 1, 2, 3
on 2D lattices, prove their topological properties, and identify minimal models.

PHASE 1 (Month 1): Build tight-binding infrastructure
- Implement Haldane model on honeycomb lattice
- Code exact Berry curvature calculation using finite differences
- Validate against known result: C = 1 for specific parameter regime

PHASE 2 (Months 1-2): Compute topological invariants
- Implement Fukui-Hatsugai-Suzuki method for exact Chern number (lattice gauge theory)
- Verify discretization error < 1e-10
- Test on Qi-Wu-Zhang model, square lattice with flux

PHASE 3 (Months 2-3): Calculate edge states
- Construct ribbon geometry with open boundaries
- Diagonalize ribbon Hamiltonian to find edge mode spectrum
- Verify bulk-edge correspondence: #(edge states) = |C|

PHASE 4 (Months 3-4): Systematic space group classification
- Enumerate all symmetry-allowed tight-binding terms for p4 (square) space group
- Scan parameter space to find regions with C 0
- Generate database of 20+ topological phases

PHASE 5 (Months 4-5): Find minimal models
- For each C {1,2,3}, find tight-binding model with fewest orbitals
- Prove lower bounds using K-theory and index theorems
- Explicitly check that fewer orbitals cannot achieve target C

PHASE 6 (Months 5-6): Export certificates
- For each model, generate TopologicalCertificate with:
  * Exact Chern number (integer)
  * Berry curvature field F(k)
  * Edge state spectrum
  * Symmetry and minimality proofs
- Export as JSON database with all verification data

SUCCESS CRITERIA:
- MVR: Haldane and Qi-Wu-Zhang models reproduced with exact C = 1
- Strong: Database of 30+ models with |C| 3, all edge states verified
- Publication: Complete classification for p1, p2, p4, p6 space groups + minimal model proofs

VERIFICATION:
- All Chern numbers computed exactly (Fukui method, no rounding)
- Edge state count verified via bulk-edge correspondence
- Symmetry constraints checked using group theory
- Minimal models proven via K-theory obstructions

Use symbolic math (SymPy) for exact arithmetic. Never use experimental databases or DFT.
All discoveries must be certificate-based and mathematically rigorous.
```

0.5 5. Success Criteria

0.5.1 Minimum Viable Result (MVR)

Within 1-2 months, the system should:

- **Canonical Models Reproduced**:
- Haldane model: C = 1 verified for M < 33 t sin()
- Qi-Wu-Zhang model: C = ±1 in topological phase
- Square lattice with -flux: C = 1
- **Berry Curvature Infrastructure**:
- Exact computation of F(k) using finite differences
- Chern number integration with error < 1e-6
- Fukui-Hatsugai-Suzuki lattice method implemented
- **Edge States**:
- Ribbon geometry for Haldane model shows 1 chiral edge mode
- Localization length verified (exponential decay into bulk)

Deliverable: $\texttt{topological}_{b}ootstrap.pywith3validatedmodels + certificates$

0.5.2 Strong Result

Within 3-4 months, add:

- **Systematic Classification:**
- Complete enumeration of C = 1,2,3 models on square lattice
- Space group p4 fully classified (all topological phases cataloged)
- Database of 30+ distinct tight-binding Hamiltonians
- **Minimal Models:**
- Prove: C = 1 requires minimum 2 orbitals (Haldane)
- Prove: C = 2 requires minimum 3 orbitals
- Prove: C = 3 requires minimum 4 orbitals
- Include explicit K-theory obstruction proofs
- **Symmetry Analysis:**

- Automatic detection of space group from Hamiltonian
- Proof that time-reversal forbids C 0
- Rotation symmetry constraints on allowed Chern numbers

Metrics:

- 30+ topological phases discovered and certified
- All Chern numbers verified exactly (integer with error < 1e-10)
- Minimal models proven via group cohomology

0.5.3 Publication-Quality Result

Within 5-6 months, achieve:

- **Complete 2D Classification**:
- All 17 wallpaper groups analyzed for allowed topology
- Database of 100+ models covering |C| 5
- Novel topological phases not in experimental literature
- **Higher Invariants**:
- index for time-reversal invariant topological insulators
- Winding numbers for 1D SSH chains
- 3D Weyl semimetal Hamiltonians with monopole charges
- **Predictive Power**:
- Identify "simple" models likely to exist in cold atom experiments
- Propose photonic crystal designs realizing C = 2,3 phases
- Predict new quantum Hall plateaus in twisted bilayer systems
- **Formal Verification**:
- Translate Chern number calculation to Lean/Isabelle
- Formally verify bulk-edge correspondence theorem
- Machine-checkable proofs of minimal model theorems

Publication Potential:

- "Pure-Thought Discovery of Topological Band Structures"
- "Minimal Tight-Binding Models for Chern Insulators: A Complete Classification"
- "Symmetry-Protected Topology: A Database Without Materials"

Impact: Establishes computational topology as predictive tool, guides experiments toward high-probability targets.

0.6 6. Verification Protocol

0.6.1 Automated Checks

```python
def verify_topological_certificate(cert: TopologicalCertificate) -> bool:
    """
    Verify all claims in certificate using independent calculations.
    """
    checks_passed = []

    # Check 1: Recompute Chern number independently
    C_recomputed = compute_chern_number_exact(
        cert.model.hamiltonian,
        list(range(cert.model.filling))
    )
    checks_passed.append(('Chern number', C_recomputed == cert.chern_number))

    # Check 2: Verify band gap
    gap = compute_band_gap(cert.model.hamiltonian, cert.model.filling)
    checks_passed.append(('Band gap > 0', gap > 0))
    checks_passed.append(('Gap matches certificate', abs(gap - cert.gap_minimum) < 1e-6))

    # Check 3: Edge state count via bulk-edge correspondence
    edge_states = identify_edge_states(cert.edge_spectrum)
    expected_edge_count = abs(cert.chern_number)
    actual_edge_count = len(edge_states)
    checks_passed.append(('Bulk-edge correspondence',
                          actual_edge_count == expected_edge_count))

    # Check 4: Symmetry constraints
    if has_time_reversal(cert.model.symmetry.point_group):
        checks_passed.append(('Time-reversal', cert.chern_number == 0))

    # Check 5: Berry curvature integration
    F_integrated = integrate_berry_curvature(cert.berry_curvature)
    checks_passed.append(('Berry curvature integral',
                          abs(F_integrated - cert.chern_number * 2*np.pi) < 1e-4))

    print("Certificate Verification:")
    for name, passed in checks_passed:
        status = "    PASS" if passed else "    FAIL"
        print(f"  {status}: {name}")

    return all(passed for _, passed in checks_passed)
```

0.6.2 Cross-Validation

- **Topological Quantum Chemistry**: Compare with Bilbao Crystallographic Server classifications
- **Literature Models**: Reproduce Bernevig-Hughes-Zhang (BHZ) model for HgTe

- **Numerical DMRG**: For 1D models, verify edge states using tensor networks

0.6.3 Exported Artifacts

For each model:

- **Model Specification JSON**:

```
{
  "model_id": "haldane_honeycomb_C1",
  "lattice": "honeycomb",
  "num_orbitals": 2,
  "parameters": {"t": 1.0, "t2": 0.2, "phi": 0.3, "M": 0.1},
  "space_group": 191,
  "chern_number": 1
}
```

- **Certificate JSON** (shown earlier)
- **Proof Document** (LaTeX with symmetry analysis)
- **Visualization**: Band structure plots, Berry curvature heatmaps, edge state wavefunction

---

0.7 7. Resources Milestones

0.7.1 Key References

Topological Band Theory:

- Haldane (1988): "Model for a Quantum Hall Effect without Landau Levels"
- Kane Mele (2005): " Topological Order and the Quantum Spin Hall Effect"
- Bernevig Hughes (2013): "Topological Insulators and Topological Superconductors"

Computational Methods:

- Fukui, Hatsugai, Suzuki (2005): "Chern Numbers in Discretized Brillouin Zone"
- Soluyanov Vanderbilt (2011): "Computing Topological Invariants without Inversion Symmetry"

K-Theory and Classification:

- Kitaev (2009): "Periodic Table for Topological Insulators"
- Freed Moore (2013): "Twisted Equivariant Matter"

0.7.2 Common Pitfalls

- **Discretization Errors**:
- Problem: Chern number is integer but numerical integration gives C = 1.03
- Solution: Use Fukui lattice gauge method, not naive integration
- **Gauge Ambiguity**:
- Problem: Berry connection jumps due to gauge choice
- Solution: Use gauge-invariant Berry curvature or smooth gauge
- **Band Crossings**:
- Problem: Bands touch, violating gap assumption
- Solution: Add small symmetry-breaking term or avoid crossing regions
- **Edge State Identification**:
- Problem: Bulk states leaking into gap region
- Solution: Increase ribbon width, use localization measure

0.7.3 Milestone Checklist

Month 1: Haldane model implemented, C = 1 verified
Month 2: Fukui method working, edge states for 3 models
Month 3: Space group p4 classification complete (10+ models)
Month 4: Minimal models for C = 1,2,3 found and proven
Month 5: Database of 50 models exported with certificates
Month 6: All 17 wallpaper groups analyzed, publication draft

0.8 8. Extensions and Open Questions

0.8.1 Immediate Extensions

- **3D Topological Insulators**: Extend to 3D with invariants (strong vs weak TI)
- **Topological Semimetals**: Weyl and Dirac points, nodal line semimetals
- **Crystalline Topological Phases**: Mirror Chern numbers, glide symmetries

0.8.2 Research Frontiers

- **Fragile Topology**: Non-stable topological phases requiring specific filling
- **Higher-Order Topology**: Corner and hinge states in 2D/3D
- **Interacting Topology**: Can pure-thought methods handle fractional Chern insulators?

0.8.3 Long-Term Vision

Build a **Topological Materials Database** purely from symmetry+computation:

- Predict new phases before experimental discovery
- Guide quantum simulator design (cold atoms, photonics, circuits)
- Provide blueprints for topological quantum computers

End of PRD 09