PRD 15: Topological Quantum Chemistry: Complete Band Structure Classification

Pure Thought AI Challenge 15

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: 6-9 months
Difficulty: High
Prerequisites: Group theory, representation theory, K-theory, crystallography, band theory

---

0.1 1. Problem Statement

0.1.1 Scientific Context

Topological Quantum Chemistry (TQC) provides a complete classification of all possible band structures for a given crystal structure, combining:

- **Space Group Symmetry**: 230 3D space groups determine allowed band representations
- **Elementary Band Representations (EBRs)**: Building blocks from which all bands can be constructed
- **Topological Indices**: Compatibility relations and symmetry indicators detect topology
- **Completeness**: Every band structure is either:
- **Atomic insulator**: Linear combination of EBRs (trivial)
- **Topological insulator**: Cannot be written as EBR sum
- **Semimetal**: Unavoidable band crossings

Key Insights:

- Each Wyckoff position (high-symmetry site) with orbital angular momentum generates an EBR
- Compatibility relations: How irreps at high-symmetry points must connect along paths
- **Fragile topology**: Bands that are topological but become trivial when adding trivial bands

Applications:

- Automated topology detection without computing Berry curvature
- Materials prediction: Given crystal structure → predict if topological
- Classification: Complete catalog of all possible topological phases for each space group

0.1.2 Core Question

Can we implement the full topological quantum chemistry framework to classify ALL band structures for a given space group using ONLY group theory and representation theory—no DFT or experimental data?

Specifically:

- Given space group G and set of occupied bands, determine if topological

- Compute symmetry indicators from irrep decomposition at high-symmetry points
- Check compatibility relations along high-symmetry paths
- Enumerate all possible topological phases (stable + fragile)
- Generate minimal tight-binding models for each topological class
- Certify results using K-theory and cohomology

0.1.3 Why This Matters

Theoretical Impact:

- Completes topological classification beyond 10-fold way
- Connects topology to standard crystallography
- Provides algorithm for exhaustive materials search

Practical Benefits:

- Predict topology of materials without expensive calculations
- Guide experimental discovery toward novel topological phases
- Database generation: catalog all 200,000 known materials by topology

Pure Thought Advantages:

- Symmetry indicators are purely algebraic (character tables)
- Compatibility relations follow from group theory
- EBR decomposition is linear algebra
- No material parameters needed—just crystal structure + filling

---

0.2 2. Mathematical Formulation

0.2.1 Problem Definition

Elementary Band Representation (EBR):

An EBR is induced from a localized Wannier orbital at a Wyckoff position q with site-symmetry group G$_q andorbitalirrep$:

```
EBR[q,   ] = Ind_{G_q}^G (  )
```

This defines a vector bundle over the Brillouin zone with specific irrep content at each k-point.

Band Representation (BR):

Any physical band structure B decomposes as:

```
B =   _i   n_i  EBR_i
```

where n$_i(canbenegativefor"subtracting"bands)$.
Topological Diagnosis:

- **Atomic Insulator**: B = ni EBRi with all n$_i$0
- **Stable Topological**: Cannot write B as non-negative EBR sum (obstruction in K-theory)
- **Fragile Topological**: B is topological, but B + (trivial bands) becomes atomic

Symmetry Indicator Group:

```
X^{BS} = {Band Structures} / {Atomic Insulators}
```

This abelian group classifies topological phases. Computed from:

```
0     X^{BS}     {Irrep vectors at K-points}     {Compatibility
    relations}     0
```

Certificate: Given band structure B (as list of irreps at high-symmetry points), compute:

- **Indicator vector**: z X$^{BS}(elementsofsymmetryindicatorgroup)$
- **EBR decomposition**: If z = 0, find B = ni EBRi with n$_i$0
- **Topology type**: Stable, fragile, or trivial
- **Minimal model**: Tight-binding H(k) realizing B

0.2.2 Input/Output Specification

Input:

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

class BandStructure:
    space_group: int  # 1-230
    dimension: int  # 2D or 3D

    # Irrep content at high-symmetry points
    irreps_at_kpoints: Dict[str, List[str]]  # {k-point name: [irrep1,
        irrep2, ...]}

    # Or: tight-binding Hamiltonian
    hamiltonian: Optional[Callable[[np.ndarray], np.ndarray]]
    filling: Optional[int]  # Number of occupied bands
```

Output:

```python
class TQCCertificate:
    band_structure: BandStructure

    # EBR analysis
    ebr_decomposition: Dict[str, int]  # {EBR_name: coefficient n_i}
    is_atomic_insulator: bool

    # Topology classification
    symmetry_indicator: Tuple[int, ...]  # z     X^{BS} (mod appropriate integers)
    topology_type: str  # "trivial", "stable_topological", "fragile_topological", "semimetal"

    # Detailed analysis
    irrep_compatibility: Dict[str, bool]  # Check each path
    compatibility_violations: List[str]  # Which paths force band crossings

    # K-theory data
    k_theory_class: Optional[str]  # Element of K-theory group
    obstruction_to_atomic: Optional[str]  # Why can't be atomic insulator

    # Minimal model
    tight_binding_model: Optional[Callable]  # Constructed H(k) if possible
    wannier_centers: Optional[List[np.ndarray]]  # Positions of localized orbitals

    proof_of_classification: str  # Mathematical derivation
```

0.3 3. Implementation Approach

0.3.1 Phase 1: Space Group and Representations (Months 1-2)

Build infrastructure for space group representation theory:

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

def load_space_group_data(sg_number: int) -> dict:
    """
    Load space group data: generators, Wyckoff positions, character tables.

    Uses crystallographic databases (Bilbao, etc.) or generates from scratch.
    """
    # For now, hardcode common space groups
    # Full implementation would parse International Tables

    if sg_number == 1:  # P1 (triclinic, most general)
        return {
            'name': 'P1',
            'point_group': 'C1',
            'generators': [np.eye(3)],
            'wyckoff_positions': ['1a'],
            'high_sym_points': {'Gamma': np.array([0, 0, 0])}
        }
    elif sg_number == 221:  # Pm-3m (simple cubic)
        return {
            'name': 'Pm-3m',
            'point_group': 'O_h',
            'generators': [  # Rotations, reflections
                rotation_matrix([1, 0, 0], np.pi/2),
                reflection_matrix([1, 0, 0])
            ],
            'wyckoff_positions': ['1a', '1b', '3c', '6d', '8e', '12f', '24g'],
            'high_sym_points': {
                'Gamma': np.array([0, 0, 0]),
                'X': np.array([np.pi, 0, 0]),
                'M': np.array([np.pi, np.pi, 0]),
                'R': np.array([np.pi, np.pi, np.pi])
            }
        }
    # ... (implement all 230 space groups)

def get_little_group_irreps(k_point: np.ndarray, space_group: int) -> List[str]:
    """
    Get irreducible representations at k-point for given space group.

    Uses character tables for little group G_k.
    """
    sg_data = load_space_group_data(space_group)
    point_group = sg_data['point_group']

    # Find little group (subgroup leaving k invariant)
    little_group = find_little_group(k_point, point_group)

    # Load character table
    char_table = get_character_table(little_group)

    # Irrep names
    irrep_names = list(char_table.keys())

    return irrep_names

def compute_ebr(wyckoff: str, orbital_irrep: str, space_group: int) -> Dict[str, List[str]]:
    """
    Compute Elementary Band Representation for Wyckoff position + orbital.

    Returns irrep content at all high-symmetry k-points.

    EBR = Ind_{G_q}^G (   )
    """
    sg_data = load_space_group_data(space_group)

    # Site symmetry at Wyckoff position
    site_symmetry = get_site_symmetry(wyckoff, space_group)

    # Induce representation from site to full group
    induced_irreps = {}

    for k_name, k_point in sg_data['high_sym_points'].items():
        # Decompose induced rep at k-point
        irreps_k = induce_representation(orbital_irrep, site_symmetry,
                                         k_point, space_group)
        induced_irreps[k_name] = irreps_k

    return induced_irreps

def induce_representation(rho: str, G_q: str, k_point: np.ndarray,
                          space_group: int) -> List[str]:
    """
    Induce representation from site group to little group at k.

    Uses Frobenius reciprocity and character orthogonality.
    """
    # Get character of rho in G_q
    char_rho = get_character(rho, G_q)

    # Little group at k
    G_k = find_little_group(k_point, space_group)

    # Induced character:  _ind (g) = (1/|G_q|)  _ {h     G: hgh^{-1}
          G_q}  _rho (hgh^{-1})
    char_induced = {}

    for g in G_k:
        char_sum = 0
        for h in get_coset_reps(G_k, G_q):
            conjugate = h @ g @ np.linalg.inv(h)
            if is_in_group(conjugate, G_q):
                char_sum += char_rho[conjugate]

        char_induced[g] = char_sum / len(G_q)

    # Decompose induced character into irreps of G_k
    irreps = decompose_character(char_induced, G_k)

    return irreps
```

Validation: Reproduce known EBRs for simple space groups (e.g., SG 221).

0.3.2 Phase 2: Compatibility Relations (Months 2-4)

Implement compatibility checking along high-symmetry paths:

```python
def get_high_symmetry_paths(space_group: int) -> List[Tuple[str, str]]:
    """
    Get standard high-symmetry paths in BZ.

    Returns list of (start_point, end_point) pairs.
    """
    sg_data = load_space_group_data(space_group)

    if sg_data['name'] == 'Pm-3m':  # Cubic
        return [
            ('Gamma', 'X'),
            ('X', 'M'),
            ('M', 'Gamma'),
            ('Gamma', 'R'),
            ('R', 'X')
        ]
    # ... (for each space group)

def check_compatibility(irreps_start: List[str], irreps_end: List[str],
                        path: Tuple[str, str], space_group: int) -> bool:
    """
    Check if irreps at start and end of path are compatible.

    Compatibility: irreps must connect via allowed subductions.
    """
    # Get compatibility matrix C[irrep_start][irrep_end]
    # C_ij = 1 if irrep_i at start can connect to irrep_j at end

    compat_matrix = get_compatibility_matrix(path, space_group)

    # Check if given irreps satisfy compatibility
    for irr_s in irreps_start:
        has_connection = False
        for irr_e in irreps_end:
            if compat_matrix[irr_s][irr_e] == 1:
                has_connection = True
                break

        if not has_connection:
            return False  # Incompatible

    return True

def get_compatibility_matrix(path: Tuple[str, str], space_group: int)
    -> Dict:
    """
    Compute compatibility matrix between irreps along path.

    Uses subduction: how irrep at high-symmetry point k decomposes
    when restricted to lower-symmetry points along path.
    """
    start, end = path

    sg_data = load_space_group_data(space_group)
    k_start = sg_data['high_sym_points'][start]
    k_end = sg_data['high_sym_points'][end]

    # Irreps at start and end
    irreps_start = get_little_group_irreps(k_start, space_group)
    irreps_end = get_little_group_irreps(k_end, space_group)

    compat = {irr_s: {irr_e: 0 for irr_e in irreps_end} for irr_s in
        irreps_start}

    # Compute subduction for each irrep at start
    for irr_s in irreps_start:
        # As we move along path, little group changes
        # Irrep decomposes into irreps of subgroup

        subduced_irreps = subduce_along_path(irr_s, k_start, k_end,
           space_group)

        for irr_e in subduced_irreps:
            if irr_e in irreps_end:
                compat[irr_s][irr_e] = 1

    return compat

def subduce_along_path(irrep: str, k_start: np.ndarray, k_end:
   np.ndarray,
                       space_group: int) -> List[str]:
    """
    Subduce irrep from k_start to k_end along straight path.
    """
    # Little groups at start and end
    G_start = find_little_group(k_start, space_group)
    G_end = find_little_group(k_end, space_group)

    # G_end     G_start typically (symmetry lowers along path)
    # Subduce: restrict irrep of G_start to G_end

    char_irrep = get_character(irrep, G_start)

    # Restrict to G_end
    char_restricted = {g: char_irrep[g] for g in G_end}

    # Decompose
    subduced = decompose_character(char_restricted, G_end)

    return subduced
```

0.3.3 Phase 3: Symmetry Indicators (Months 4-5)

Compute symmetry indicator group X^{BS} :

```python
def compute_symmetry_indicator_group(space_group: int) -> dict:
    """
    Compute symmetry indicator group X^{BS} for space group.

    Returns group structure (e.g.,                  ) and generators.
    """
    sg_data = load_space_group_data(space_group)

    # Irrep vector space: for each k-point, for each irrep, count
    irrep_space_dim = 0
    for k_name, k_point in sg_data['high_sym_points'].items():
        irreps = get_little_group_irreps(k_point, space_group)
        irrep_space_dim += len(irreps)

    # Compatibility relations provide constraints
    num_constraints = 0
    for path in get_high_symmetry_paths(space_group):
        # Each path gives compatibility equations
        num_constraints += count_compatibility_constraints(path,
            space_group)

    # X^{BS} = ker(compatibility) / im(EBR)
    # Compute using Smith normal form

    # Build matrix: rows = compatibility relations, cols = irrep vectors
    compat_matrix = build_compatibility_matrix(space_group)

    # Build EBR matrix: rows = irrep vectors, cols = EBRs
    ebr_matrix = build_ebr_matrix(space_group)

    # Smith normal form to get X^{BS}
    X_BS = smith_normal_form_quotient(compat_matrix, ebr_matrix)

    return X_BS

def classify_band_structure(irreps_at_kpoints: Dict[str, List[str]],
                            space_group: int) -> TQCCertificate:
    """
    Classify band structure from irrep content.
    """
    cert = TQCCertificate()

    # Check compatibility along all paths
    paths = get_high_symmetry_paths(space_group)
    all_compatible = True

    for path in paths:
        start, end = path
        compatible = check_compatibility(
            irreps_at_kpoints[start],
            irreps_at_kpoints[end],
            path,
            space_group
        )

        cert.irrep_compatibility[path] = compatible

        if not compatible:
            all_compatible = False
            cert.compatibility_violations.append(f"{start}-{end}")

    if not all_compatible:
        cert.topology_type = "semimetal"
        cert.is_atomic_insulator = False
        return cert

    # Compute symmetry indicator
    indicator = compute_indicator_vector(irreps_at_kpoints, space_group)
    cert.symmetry_indicator = indicator

    # Try EBR decomposition
    ebr_decomp, success = decompose_into_ebrs(irreps_at_kpoints,
        space_group)

```

```python
    if success:
        cert.is_atomic_insulator = True
        cert.topology_type = "trivial"
        cert.ebr_decomposition = ebr_decomp
    else:
        cert.is_atomic_insulator = False

        # Check if fragile or stable
        is_fragile = check_fragile_topology(indicator, space_group)

        if is_fragile:
            cert.topology_type = "fragile_topological"
        else:
            cert.topology_type = "stable_topological"

    return cert

def decompose_into_ebrs(irreps: Dict[str, List[str]], space_group: int)
   -> Tuple[Dict, bool]:
    """
    Try to decompose band structure as non-negative sum of EBRs.

    Returns: (decomposition, success)
    """
    # Get all EBRs for this space group
    all_ebrs = enumerate_ebrs(space_group)

    # Set up linear system: find n_i     0 such that
    #  _i  n_i EBR_i = given irreps at each k-point

    # This is integer linear programming problem
    from scipy.optimize import linprog

    # ... (solve ILP for non-negative coefficients)

    # If solution exists with all n_i     0: atomic insulator
    # Otherwise: topological

    return decomposition, success
```

0.3.4 Phase 4: Model Construction (Months 5-7)

Build minimal tight-binding models realizing each topological class:

```python
def construct_tight_binding_from_ebrs(ebr_decomp: Dict[str, int],
                                      space_group: int) -> Callable:
    """
    Construct tight-binding Hamiltonian from EBR decomposition.

    Each EBR     Wannier orbital at specific Wyckoff position.
    """
    lattice_vectors = get_lattice_vectors(space_group)
    wyckoff_positions = get_wyckoff_positions(space_group)

    # Build H(k) from Wannier functions
    def H(k: np.ndarray) -> np.ndarray:
        H_k = np.zeros((total_orbitals, total_orbitals), dtype=complex)

        for ebr_name, coeff in ebr_decomp.items():
            if coeff <= 0:
                continue

            # Get Wannier center and orbital for this EBR
            wyckoff, orbital = parse_ebr_name(ebr_name)
            r_center = wyckoff_to_position(wyckoff, space_group)

            # Add contribution to H(k)
            H_k += coeff * wannier_contribution(k, r_center, orbital, space_group)

        return H_k

    return H

def construct_topological_model(indicator: Tuple[int, ...],
                               topology_type: str,
                               space_group: int) -> Callable:
    """
    Construct minimal tight-binding model realizing given topology.

    For each topological class, build explicit Hamiltonian.
    """
    if topology_type == "trivial":
        # Use EBR decomposition
        return construct_from_ebrs(...)

    elif topology_type == "stable_topological":
        # Build model with obstruction
        # E.g., for        TI in certain space groups

        if space_group == 230 and indicator == (1,):  # Example
            # Construct Fu-Kane-Mele        TI
            return fu_kane_model()

    elif topology_type == "fragile_topological":
        # Fragile models require specific constructions
        return construct_fragile_model(indicator, space_group)

    return None
```

0.3.5 Phase 5: Complete Classification (Months 7-8)

Enumerate all topological phases for each space group:

```
def classify_all_phases_for_space_group(sg: int, max_bands: int = 10) -> List:
    """
    Enumerate all distinct topological phases for space group sg.

    Up to max_bands occupied bands.
    """
    phases = []
```

```python

    # Get X^{BS} structure
    indicator_group = compute_symmetry_indicator_group(sg)

    # For each element of indicator group
    for indicator in enumerate_group_elements(indicator_group):
        # Check if realizable with     max_bands
        realizable, model = try_construct_model(indicator, sg,
            max_bands)

        if realizable:
            cert = generate_tqc_certificate(model)

            phases.append({
                'space_group': sg,
                'indicator': indicator,
                'topology_type': cert.topology_type,
                'min_bands': compute_min_bands(indicator, sg),
                'model': model
            })

    return phases

def generate_tqc_database(space_group_list: List[int]) -> dict:
    """
    Generate complete TQC database for given space groups.
    """
    database = {'space_groups': {}}

    for sg in space_group_list:
        print(f"Classifying space group {sg}...")

        phases = classify_all_phases_for_space_group(sg, max_bands=10)

        database['space_groups'][sg] = {
            'name': get_space_group_name(sg),
            'indicator_group':
                str(compute_symmetry_indicator_group(sg)),
            'num_phases': len(phases),
            'phases': phases
        }

    return database
```

0.3.6 Phase 6: Material Prediction (Months 8-9)

Apply TQC to predict topology of real materials (from crystal structure only):

```python
def predict_material_topology(crystal_structure: dict) ->
   TQCCertificate:
    """
    Predict if material is topological from crystal structure alone.

    Input: crystal structure (space group + atomic positions + orbitals)
    Output: TQC certificate with topology classification
    """
    space_group = crystal_structure['space_group']
    atoms = crystal_structure['atoms']  # [(element, wyckoff,
        orbitals), ...]

    # Build band representation from atomic orbitals
    band_rep = {}

    for element, wyckoff, orbitals in atoms:
        for orbital in orbitals:  # s, p, d, etc.
            # Get EBR for this Wyckoff + orbital
            ebr = compute_ebr(wyckoff, orbital, space_group)

            # Add to total band representation
            band_rep = add_band_representations(band_rep, ebr)

    # Classify
    cert = classify_band_structure(band_rep, space_group)

    return cert

def screen_materials_database(materials: List[dict]) -> List[dict]:
    """
    Screen materials database for topological candidates.

    Input: list of crystal structures
    Output: list of materials with non-trivial topology
    """
    topological_materials = []

    for material in materials:
        cert = predict_material_topology(material)

        if cert.topology_type in ["stable_topological",
            "fragile_topological"]:
            topological_materials.append({
                'name': material['name'],
                'formula': material['formula'],
                'space_group': material['space_group'],
                'topology': cert.topology_type,
                'indicator': cert.symmetry_indicator,
                'certificate': cert
            })

    return topological_materials
```

0.4 4. Example Starting Prompt

```
You are a mathematician specializing in topological quantum chemistry.
    Implement the complete TQC
framework to classify ALL band structures for space group 221 (Pm-3m)
    using ONLY group theory.

```

```
OBJECTIVE: Build EBR database, compute symmetry indicators, classify
   all topological phases.

PHASE 1 (Months 1-2): Space group infrastructure
- Load SG 221 data: Wyckoff positions (1a, 1b, 3c, ...), high-sym
   points ( , X, M, R)
- Implement character tables for point group O_h
- Compute EBRs for all Wyckoff + orbital combinations
- Verify: EBR[1a, s] gives specific irreps at  , X, M, R

PHASE 2 (Months 2-4): Compatibility relations
- Build compatibility matrices for paths:  -X, X-M, M-  ,  -R
- Implement subduction: how irreps decompose along paths
- Check: specific irrep combinations force band crossings

PHASE 3 (Months 4-5): Symmetry indicators
- Compute X^{BS} =      ^3 for SG 221 (known result)
- Implement indicator vector calculation from irrep content
- Classify: given band structure    ( z  ,  z  ,  z  )

PHASE 4 (Months 5-7): Model construction
- For each of 8 elements of       , build minimal tight-binding model
- Verify irrep content matches target indicator
- Compute band structures, check gaps

PHASE 5 (Months 7-8): Complete classification
- Enumerate all topological phases for SG 221
- Identify: trivial (z=0), 7 topological classes
- Document minimal band numbers for each phase

PHASE 6 (Months 8-9): Material predictions
- Apply to perovskite structures (SG 221)
- Predict: which are topological based on atomic orbitals
- Generate candidate list for experimental verification

SUCCESS CRITERIA:
- MVR: EBR database for SG 221, compatibility checks working
- Strong: All 8 topological phases classified, models constructed
- Publication: Material predictions, verification against known TIs

VERIFICATION:
- X^{BS} =        (matches literature for SG 221)
- 8 distinct topological phases identified
- Tight-binding models reproduce target indicators
- Predictions cross-checked against Materials Project

Pure group theory + linear algebra. No DFT.
All results certificate-based with exact indicator computation.
```

0.5 5. Success Criteria

MVR (2-3 months): EBR database for 1 space group, compatibility checks

Strong (5-7 months): Complete classification of 1 space group, models built

Publication (8-9 months): Multi-space-group database, material predictions

---

0.6 6. Verification Protocol

Cross-check against:

- Bilbao Crystallographic Server
- Topological Materials Database
- Published TQC results

0.7 7. Resources Milestones

References:

- Bradlyn et al. (2017): "Topological Quantum Chemistry"
- Po, Watanabe, Vishwanath (2017): "Symmetry-Based Indicators of Band Topology"
- Vergniory et al. (2019): "A Complete Catalogue of High-Quality Topological Materials"

Milestones:

- Month 2: EBR database for SG 221
- Month 5: X$^{BS} computed, indicators working$
- Month 7: All phases classified
- Month 9: Material predictions complete

0.8 8. Extensions

- **Magnetic Space Groups**: 1651 groups including magnetic order
- **Higher-Order TQC**: Connecting to higher-order topology
- **Interacting Systems**: Generalization to strongly correlated materials

Long-Term Vision: Automated topological materials discovery—input crystal structure, output predicted topology. No experiments or simulations needed until final verification stage.

End of PRD 15