PRD 14: Topological Semimetals: Weyl and Dirac Points from Symmetry

Pure Thought AI Challenge 14

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: 5-7 months
Difficulty: Medium-High
Prerequisites: Band theory, topology, group theory, differential geometry

---

0.1 1. Problem Statement

0.1.1 Scientific Context

Topological semimetals are 3D materials where conduction and valence bands touch at isolated points (Weyl/Dirac) or lines (nodal lines) in the Brillouin zone, protected by topology and symmetry.

Weyl Points:

- **Band crossing**: Two non-degenerate bands touch at momentum k_W
- **Topological charge**: ± 1 chirality (monopole of Berry curvature)
- **Fermi arc surface states**: Connect projections of opposite-chirality Weyl points
- **Requires**: Either broken time-reversal (T) or inversion (I) symmetry

Dirac Points:

- **4-fold degeneracy**: Two Weyl points of opposite chirality pinned together by symmetry
- **Requires**: Both T and I symmetry present
- **Can split**: Into Weyl pairs when symmetry broken

Nodal Lines:

- **1D band crossing**: Bands touch along continuous curves in BZ
- **Protected by**: Mirror/glide symmetries + additional symmetries
- **Drumhead surface states**: Flat bands on surface

Key Properties:

- **Nielsen-Ninomiya theorem**: Weyl points come in pairs with opposite chirality ($C_i = 0$)
- **Bulk-boundary correspondence**: Fermi arcs connect Weyl points of opposite chirality
- **Anomalies**: Chiral anomaly causes magnetoresistance, optical responses

0.1.2 Core Question

Can we systematically construct tight-binding models with Weyl/Dirac points using ONLY symmetry constraints and topology—without materials databases or DFT?

Specifically:

- Given space group G, find minimal models hosting Weyl points
- Compute exact positions k$_W$ *and chiralities* C_W
- Prove Fermi arc existence and topology
- Optimize nodal line geometries (linking, knotting)
- Certify robustness against disorder and perturbations
- Export 3D visualizations of Fermi surfaces and arcs

0.1.3 Why This Matters

Theoretical Impact:

- Completes topological classification for gapless systems
- Connects knot theory to condensed matter
- Tests bulk-boundary correspondence in 3D

Practical Benefits:

- Novel transport phenomena (chiral anomaly, nonlocal resistance)
- Topological quantum computing platforms
- Optoelectronic devices with unusual responses

Pure Thought Advantages:

- Weyl points are topological defects (no material parameters needed)
- Chirality computed from Berry curvature (exact)
- Symmetry analysis determines allowed positions
- Surface states from semi-infinite geometry (pure mathematics)

0.2 2. Mathematical Formulation

0.2.1 Problem Definition

A **Weyl point** at momentum k$_W$ *isapointwhere* :

- **Two bands touch**: E(k *W) = E(k*W)
- **Linear dispersion**: E(k) E*F* ± *v*F |k - k *W| near k*W
- **Topological charge**:

```
    C_W = (1/2  )   _  {S} F(k)    dS
```

where S is a small sphere around k$_W$, *andFisBerrycurvature*

Weyl Hamiltonian (low-energy effective):

```
H(q) =  v_F  (      q)
```

where q = k - k *W, are Pauli matrices, chirality C*W = sign(det(v$_F$))

Dirac Hamiltonian (4×4):

```
H(q) =  v_F  (      q)
```

where are 4×4 gamma matrices (two copies of Pauli matrices)

Nodal Line: 1D curve (t) in BZ where bands touch, characterized by:

- linking invariant (how nodal lines link)
- **Drumhead** flat surface state filling interior of projected loop

0.2.2 Certificate Requirements

- **Weyl Point Certificate**: Exact (k *W, C*W) pairs
- **Fermi Arc Topology**: Connectivity of arcs on surface BZ
- **Symmetry Protection**: Proof that Weyl/Dirac points cannot gap without breaking symmetry
- **Nodal Line Geometry**: Linking numbers, knot invariants
- **Surface State Verification**: Existence of arcs/drumheads

0.2.3 Input/Output Specification

Input:

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

class SemimetalModel:
    dimension: int  # Must be 3D
    space_group: int
    time_reversal: bool  # T symmetry
    inversion: bool  # I symmetry
```

```
    hamiltonian: Callable[[np.ndarray], np.ndarray]  # H(k), 3D momentum
    num_bands: int
```

Output:

```
class SemimetalCertificate:
    model: SemimetalModel

    # Weyl points
    weyl_points: List[Tuple[np.ndarray, int]]  # [(k_W, chirality), ...]
    total_chirality: int  # Should be 0 (Nielsen-Ninomiya)

    # Dirac points
    dirac_points: List[np.ndarray]  # [k_D, ...]
    dirac_splitting: Optional[List[Tuple]]  # How Dirac     2 Weyl when
        symmetry broken

    # Nodal lines
    nodal_lines: List[Callable]  # [     (t),      (t), ...]
        parameterized curves
    linking_numbers: np.ndarray  # L_ij = linking of  _i ,  _j
    knot_invariants: List[str]  # Alexander polynomial, etc.

    # Surface states
    fermi_arcs: List[Tuple[np.ndarray, np.ndarray]]  # [(k_start,
        k_end), ...] on surface
    drumhead_states: Optional[np.ndarray]  # For nodal lines

    # Verification
    berry_curvature_field: Callable[[np.ndarray], np.ndarray]  # F(k)
    surface_spectral_function: Callable[[np.ndarray], float]  # A(k,
        E=0)

    proof_of_topology: str  # Mathematical derivation
```

0.3 3. Implementation Approach

0.3.1 Phase 1: Minimal Weyl Semimetal Model (Months 1-2)

Implement simplest Weyl semimetal (broken inversion):

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

def minimal_weyl_model(m: float, b: float) -> Callable:
    """
    Minimal 2-band model hosting a pair of Weyl points.

    H(k) = (b k_z + m)  _x  + b k_x  _y  + b k_y  _z

    Weyl points at k_W =   (0, 0, m/b) with opposite chirality.

    Breaks inversion symmetry (due to k_z term).
    """
```

```
    def H(k: np.ndarray) -> np.ndarray:
        kx, ky, kz = k[0], k[1], k[2]

        sx = np.array([[0, 1], [1, 0]])
        sy = np.array([[0, -1j], [1j, 0]])
        sz = np.array([[1, 0], [0, -1]])

        H_k = (b*kz + m)*sx + b*kx*sy + b*ky*sz

        return H_k

    return H

def find_band_crossings(H_func: Callable, k_range: Tuple[float, float, float],
                        N_k: int = 50) -> List[np.ndarray]:
    """
    Find points in BZ where bands touch (gap closes).

    Returns list of k-points where min|E_i - E_j| < threshold.
    """
    kx_vals = np.linspace(-k_range[0], k_range[0], N_k)
    ky_vals = np.linspace(-k_range[1], k_range[1], N_k)
    kz_vals = np.linspace(-k_range[2], k_range[2], N_k)

    crossing_points = []

    for kx in kx_vals:
        for ky in ky_vals:
            for kz in kz_vals:
                k = np.array([kx, ky, kz])
                evals = np.linalg.eigvalsh(H_func(k))

                # Check for near-degeneracy
                gaps = [abs(evals[i+1] - evals[i]) for i in range(len(evals)-1)]
                min_gap = min(gaps)

                if min_gap < 1e-3:  # Threshold for crossing
                    crossing_points.append(k)

    return crossing_points

def refine_weyl_point(H_func: Callable, k_initial: np.ndarray,
                      tol: float = 1e-10) -> np.ndarray:
    """
    Refine Weyl point position to machine precision.

    Minimize gap = | E  (k) -  E  (k)| around initial guess.
    """
    from scipy.optimize import minimize

    def gap_function(k):
        evals = np.linalg.eigvalsh(H_func(k))
        return min([abs(evals[i+1] - evals[i]) for i in range(len(evals)-1)])
```

```
    result = minimize(gap_function, k_initial, method='Powell', tol=tol)

    return result.x
```

Validation: Reproduce textbook Weyl model, verify $k_W positions$.

0.3.2 Phase 2: Chirality Computation (Months 2-3)

Compute topological charge of each Weyl point:

```
def berry_curvature_3d(H_func: Callable, k: np.ndarray,
                       band_idx: int, delta: float = 1e-5) ->
                          np.ndarray:
    """
    Compute Berry curvature F = (F_x, F_y, F_z) at k.

    F_   =   _ {      }    _    A_

    Returns 3-vector.
    """
    # Get wavefunction
    evals, evecs = eigh(H_func(k))
    sorted_idx = np.argsort(evals)
    u_k = evecs[:, sorted_idx[band_idx]]

    F = np.zeros(3)

    # F_x =   _y   A_z -   _z   A_y
    # Use finite differences

    dk_y = np.array([0, delta, 0])
    dk_z = np.array([0, 0, delta])

    # A_z at k and k+ y
    A_z_k = berry_connection_component(H_func, k, band_idx,
       direction=2, delta=delta)
    A_z_k_plus_y = berry_connection_component(H_func, k+dk_y, band_idx,
       direction=2, delta=delta)

    # A_y at k and k+ z
    A_y_k = berry_connection_component(H_func, k, band_idx,
       direction=1, delta=delta)
    A_y_k_plus_z = berry_connection_component(H_func, k+dk_z, band_idx,
       direction=1, delta=delta)

    F[0] = (A_z_k_plus_y - A_z_k)/delta - (A_y_k_plus_z - A_y_k)/delta

    # Similarly for F_y and F_z
    # ... (analogous calculations)

    return F

def compute_weyl_chirality(H_func: Callable, k_W: np.ndarray,
                           radius: float = 0.1, N_theta: int = 20,
                              N_phi: int = 20) -> int:
```

```
    """
    Compute chirality C_W =     F    dS around Weyl point.

    Integrate Berry curvature over sphere of radius r around k_W.
    """
    # Parameterize sphere: k = k_W + r(sin    cos    , sin    sin    ,
       cos    )
    theta_vals = np.linspace(0, np.pi, N_theta)
    phi_vals = np.linspace(0, 2*np.pi, N_phi)

    flux = 0

    for theta in theta_vals:
        for phi in phi_vals:
            # Point on sphere
            k = k_W + radius * np.array([
                np.sin(theta)*np.cos(phi),
                np.sin(theta)*np.sin(phi),
                np.cos(theta)
            ])

            # Berry curvature
            F = berry_curvature_3d(H_func, k, band_idx=0)  # Lower band

            # Outward normal
            n = np.array([np.sin(theta)*np.cos(phi),
                          np.sin(theta)*np.sin(phi),
                          np.cos(theta)])

            # F    dS (with Jacobian for spherical measure)
            dS = radius**2 * np.sin(theta) *
                (theta_vals[1]-theta_vals[0]) * (phi_vals[1]-phi_vals[0])
            flux += np.dot(F, n) * dS

    # Chirality is flux/(2  )
    chirality = int(np.round(flux / (2*np.pi)))

    return chirality
```

0.3.3 Phase 3: Fermi Arc Calculation (Months 3-4)

Compute surface states and Fermi arcs:

```
def surface_green_function(H_bulk: Callable, k_parallel: np.ndarray,
                           energy: float = 0, surface_normal: str = 'z',
                           N_layers: int = 100) -> np.ndarray:
    """
    Compute surface Green's function G( k _  , E) for semi-infinite
       geometry.

    Uses iterative method (transfer matrix or recursive Green's
       function).
    """
    # For simplicity, use slab geometry with large N_layers

```

```
    H_slab = construct_slab_hamiltonian(H_bulk, k_parallel,
        surface_normal, N_layers)

    # Green's function: G = (E - H + i )^{-1}
    eta = 1e-3  # Small imaginary part
    dim = H_slab.shape[0]

    G = np.linalg.inv((energy + 1j*eta)*np.eye(dim) - H_slab)

    # Project onto surface layer
    num_orbitals = H_bulk(np.array([0, 0, 0])).shape[0]
    G_surface = G[:num_orbitals, :num_orbitals]

    return G_surface

def compute_surface_spectral_function(H_bulk: Callable, k_parallel:
    np.ndarray,
                                      energy: float = 0) -> float:
    """
    Surface spectral function A( k _  , E) = -Im Tr G( k _  , E).

    Peaks indicate surface states.
    """
    G_surf = surface_green_function(H_bulk, k_parallel, energy)

    A = -np.imag(np.trace(G_surf))

    return A

def map_fermi_arcs(H_bulk: Callable, weyl_points: List[Tuple],
                   N_k: int = 100) -> List[Tuple]:
    """
    Map Fermi arcs connecting Weyl point projections on surface BZ.

    Returns list of arcs as (k_start, k_end) pairs.
    """
    # Project Weyl points onto surface (kz=0 plane)
    weyl_projections = [(np.array([k[0], k[1]]), chi) for k, chi in
        weyl_points]

    # Compute spectral function on surface BZ at E=0
    kx_surf = np.linspace(-np.pi, np.pi, N_k)
    ky_surf = np.linspace(-np.pi, np.pi, N_k)

    spectral_map = np.zeros((N_k, N_k))

    for i, kx in enumerate(kx_surf):
        for j, ky in enumerate(ky_surf):
            k_par = np.array([kx, ky])
            spectral_map[i, j] =
                compute_surface_spectral_function(H_bulk, k_par,
                energy=0)

    # Identify arcs: high spectral weight curves connecting Weyl
        projections
    # Use image processing / contour finding
```

```
    from skimage.feature import peak_local_max

    # Find high-intensity ridges (arcs)
    arc_pixels = spectral_map > 0.5 * np.max(spectral_map)

    # Trace paths (simplifiedfull implementation needs sophisticated
       tracking)
    arcs = []
    # ... (arc tracing algorithm)

    return arcs
```

0.3.4 Phase 4: Dirac Points and Splitting (Months 4-5)

Study Dirac semimetals and symmetry breaking:

```
def dirac_semimetal_model(v_F: float = 1.0) -> Callable:
    """
    Minimal Dirac semimetal with both T and I symmetry.

    4-band model: two Weyl points pinned together at k=0.

    H(k) = v_F (k_x       + k_y       + k_z      )

    where  _i  are 4 4  Dirac matrices.
    """
    # Gamma matrices (one representation)
    Gamma1 = np.kron(np.array([[0, 1], [1, 0]]), np.eye(2))
    Gamma2 = np.kron(np.array([[0, -1j], [1j, 0]]), np.eye(2))
    Gamma3 = np.kron(np.array([[1, 0], [0, -1]]), np.array([[0, 1], [1,
       0]]))

    def H(k: np.ndarray) -> np.ndarray:
        kx, ky, kz = k[0], k[1], k[2]
        return v_F * (kx*Gamma1 + ky*Gamma2 + kz*Gamma3)

    return H

def split_dirac_into_weyl(H_dirac: Callable, breaking_term: str,
                          strength: float) -> Callable:
    """
    Split Dirac point into two Weyl points by breaking symmetry.

    breaking_term:
    - 'inversion': Add term breaking I symmetry
    - 'time_reversal': Add term breaking T symmetry (magnetic field)
    """
    def H_split(k: np.ndarray) -> np.ndarray:
        H_0 = H_dirac(k)

        if breaking_term == 'inversion':
            # Add k_z  term or constant mass
            delta_H = strength * np.kron(np.array([[1, 0], [0, -1]]),
               np.eye(2))
        elif breaking_term == 'time_reversal':
            # Add magnetic field along z
```

```
            delta_H = strength * np.kron(np.eye(2), np.array([[1, 0],
                [0, -1]]))
        else:
            delta_H = np.zeros_like(H_0)

        return H_0 + delta_H

    return H_split

def trace_dirac_to_weyl_transition(H_dirac: Callable, strength_values:
    np.ndarray):
    """
    Track how Dirac point splits into Weyl pair as symmetry-breaking
        increased.
    """
    weyl_separation = []

    for s in strength_values:
        H_split = split_dirac_into_weyl(H_dirac, 'inversion', s)

        # Find Weyl points
        weyl_pts = find_band_crossings(H_split, k_range=(np.pi, np.pi,
            np.pi))

        if len(weyl_pts) >= 2:
            # Distance between Weyl pair
            dist = np.linalg.norm(weyl_pts[0] - weyl_pts[1])
            weyl_separation.append(dist)
        else:
            weyl_separation.append(0)

    return weyl_separation
```

0.3.5 Phase 5: Nodal Lines (Months 5-6)

Design semimetals with nodal line degeneracies:

```
def nodal_line_model(mirror_plane: str = 'xy') -> Callable:
    """
    Model with nodal line protected by mirror symmetry.

    Bands touch along a circle in BZ (e.g., k_z = 0, k_x  + k_y  =
        r   ).
    """
    def H(k: np.ndarray) -> np.ndarray:
        kx, ky, kz = k[0], k[1], k[2]

        sx = np.array([[0, 1], [1, 0]])
        sy = np.array([[0, -1j], [1j, 0]])
        sz = np.array([[1, 0], [0, -1]])

        # Nodal line at kz=0, kx +ky =1
        H_k = (kx**2 + ky**2 - 1)*sx + kz*sy

        return H_k

```

```
    return H

def extract_nodal_line(H_func: Callable, N_sample: int = 1000) ->
    Callable:
    """
    Find parameterization   (t) of nodal line in BZ.

    Returns: curve   : [0, 2  ]           (momentum space)
    """
    # Sample BZ to find where gap closes
    gap_threshold = 1e-4
    nodal_points = []

    # ... (sample k-space, identify near-degeneracies)

    # Fit smooth curve through nodal points
    from scipy.interpolate import splprep, splev

    tck, u = splprep([nodal_points[:, 0], nodal_points[:, 1],
        nodal_points[:, 2]], s=0)

    def gamma(t):
        return np.array(splev(t, tck))

    return gamma

def compute_linking_number(gamma1: Callable, gamma2: Callable) -> int:
    """
    Compute Gauss linking number for two nodal lines.

    L = (1/4  )        (     '(s)          '(t))    (     (s) -
            (t)) / |     (s) -      (t)|   ds dt
    """
    # Numerical integration
    N_s, N_t = 100, 100
    s_vals = np.linspace(0, 2*np.pi, N_s)
    t_vals = np.linspace(0, 2*np.pi, N_t)

    linking = 0

    for s in s_vals:
        for t in t_vals:
            g1_s = gamma1(s)
            g2_t = gamma2(t)

            # Derivatives
            ds = s_vals[1] - s_vals[0]
            dt = t_vals[1] - t_vals[0]

            g1_prime = (gamma1(s + ds) - gamma1(s)) / ds
            g2_prime = (gamma2(t + dt) - gamma2(t)) / dt

            diff = g1_s - g2_t
            dist_cubed = np.linalg.norm(diff)**3

            if dist_cubed > 1e-6:
```

```
                integrand = np.dot(np.cross(g1_prime, g2_prime), diff)
                    / dist_cubed
                linking += integrand * ds * dt

    linking /= (4*np.pi)

    return int(np.round(linking))
```

0.3.6 Phase 6: Certification and Database (Months 6-7)

Generate complete certificates and database:

```
def generate_semimetal_certificate(model: SemimetalModel) ->
    SemimetalCertificate:
    """
    Generate complete certificate for topological semimetal.
    """
    cert = SemimetalCertificate(model=model)

    # Find Weyl points
    crossings = find_band_crossings(model.hamiltonian, k_range=(np.pi,
        np.pi, np.pi))

    weyl_points = []
    for k_cross in crossings:
        k_refined = refine_weyl_point(model.hamiltonian, k_cross)
        chirality = compute_weyl_chirality(model.hamiltonian, k_refined)

        if chirality != 0:
            weyl_points.append((k_refined, chirality))

    cert.weyl_points = weyl_points
    cert.total_chirality = sum([chi for _, chi in weyl_points])

    # Verify Nielsen-Ninomiya
    assert cert.total_chirality == 0, "Chirality sum must be zero!"

    # Fermi arcs
    cert.fermi_arcs = map_fermi_arcs(model.hamiltonian, weyl_points)

    # Nodal lines (if present)
    # ... (detect and parameterize)

    # Berry curvature field
    cert.berry_curvature_field = lambda k:
        berry_curvature_3d(model.hamiltonian, k, 0)

    return cert

def export_semimetal_visualization(cert: SemimetalCertificate,
    output_dir: Path):
    """
    Export 3D visualizations of Weyl points, Fermi arcs, nodal lines.
    """
    import matplotlib.pyplot as plt
    from mpl_toolkits.mplot3d import Axes3D
```

```
41
42      # Plot Weyl points in 3D BZ
43      fig = plt.figure()
44      ax = fig.add_subplot(111, projection='3d')
45
46      for k_W, chi in cert.weyl_points:
47          color = 'red' if chi > 0 else 'blue'
48          ax.scatter(k_W[0], k_W[1], k_W[2], c=color, s=100, marker='o')
49
50      ax.set_xlabel('kx')
51      ax.set_ylabel('ky')
52      ax.set_zlabel('kz')
53      ax.set_title('Weyl Points (red=+1, blue=-1)')
54
55      plt.savefig(output_dir / 'weyl_points_3d.png')
56
57      # Plot Fermi arcs on surface
58      # ... (2D plot of arcs connecting Weyl projections)
59
60  def generate_semimetal_database() -> dict:
61      """
62      Database of topological semimetals.
63      """
64      database = {'models': []}
65
66      # Weyl semimetals
67      for config in ['minimal', 'type-II', 'multi-weyl']:
68          model = construct_weyl_model(config)
69          cert = generate_semimetal_certificate(model)
70
71          database['models'].append({
72              'type': 'Weyl',
73              'configuration': config,
74              'num_weyl_points': len(cert.weyl_points),
75              'weyl_positions': [k.tolist() for k, _ in cert.weyl_points],
76              'certificate_path': export_certificate(cert)
77          })
78
79      # Dirac semimetals
80      # ... (similar for Dirac, nodal line models)
81
82      return database
```

0.4 4. Example Starting Prompt

```
1  You are a condensed matter theorist specializing in topological
       semimetals. Design tight-binding
2  models with Weyl/Dirac points using ONLY symmetry and topologyno DFT
       or materials databases.
3
4  OBJECTIVE: Construct minimal Weyl semimetal, compute chiralities  1 ,
       verify Fermi arcs.
5
```

```
PHASE 1 (Months 1-2): Minimal Weyl model
- Implement 2-band H(k) = (bk_z + m) _x  + bk_x _y + bk_y _z
- Find Weyl points at k_W =   (0,0,m/b)
- Refine positions to machine precision

PHASE 2 (Months 2-3): Chirality calculation
- Compute Berry curvature F(k) on sphere around each Weyl point
- Integrate   FdS   to get chirality C_W =  1
- Verify Nielsen-Ninomiya:  C_i  = 0

PHASE 3 (Months 3-4): Fermi arcs
- Construct slab geometry (semi-infinite in z)
- Compute surface Green's function G( k _  , E=0)
- Map spectral function A( k _  ) = -Im Tr G
- Identify arcs connecting Weyl projections

PHASE 4 (Months 4-5): Dirac semimetals
- Build 4-band Dirac model with T and I symmetry
- Split Dirac    2 Weyl by breaking I
- Track Weyl separation vs perturbation strength

PHASE 5 (Months 5-6): Nodal lines
- Construct model with mirror-protected nodal line
- Parameterize nodal curve   (t) in BZ
- Compute linking numbers for multi-component lines

PHASE 6 (Months 6-7): Database and visualization
- Generate certificates for 10 semimetal types
- Export 3D visualizations of BZ, Weyl points, arcs
- Classify by space group symmetry

SUCCESS CRITERIA:
- MVR: Minimal Weyl model with verified C_W =  1
- Strong: Fermi arcs computed and visualized
- Publication: Complete database + linking number calculations

VERIFICATION:
- Chirality exact: C_W    {-1, 0, +1} (integer)
- Nielsen-Ninomiya:  _i  C_i = 0
- Fermi arcs connect opposite-chirality Weyl points
- Linking numbers computed for nodal lines

Pure topology + linear algebra. No DFT.
All results certificate-based with exact chirality computation.
```

0.5 5. Success Criteria

MVR (2 months): Minimal Weyl model, chirality verified
Strong (4-5 months): Fermi arcs, Dirac splitting
Publication (6-7 months): Complete database, nodal line linking

---

0.6 6. Verification Protocol

Automated checks: chirality sum, arc connectivity, symmetry verification.

0.7 7. Resources Milestones

References:

- Wan et al. (2011): "Topological Semimetal and Fermi-Arc Surface States"
- Burkov Balents (2011): "Weyl Semimetal in a Topological Insulator Multilayer"
- Fang et al. (2016): "Topological Nodal Line Semimetals"

Milestones:

- Month 2: Weyl model validated
- Month 4: Fermi arcs mapped
- Month 6: Nodal lines classified

0.8 8. Extensions

- **Type-II Weyl**: Tilted cones
- **Hopf Nodal Links**: Knotted nodal lines
- **Non-Hermitian Semimetals**: Exceptional points

End of PRD 14