PRD 12: Topological Mechanical Metamaterials

Pure Thought AI Challenge 12

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: 4-7 months
Difficulty: Medium-High
Prerequisites: Classical mechanics, elasticity theory, graph theory, computational mechanics

---

0.1 1. Problem Statement

0.1.1 Scientific Context

Mechanical metamaterials are architected structures whose mechanical properties arise from geometry rather than composition. When combined with **topological physics**, they exhibit:

- **Topological Zero Modes**: Floppy edge modes that cost zero energy to excite
- **Protected Deformations**: Robust against disorder, defects, and manufacturing errors
- **Programmable Mechanics**: Designer elasticity, auxetic behavior, mechanical computing

The key insight is that **phononic band structures** (vibrations in elastic materials) obey the same topological principles as electronic and photonic systems.

Maxwell Counting: For a framework of N sites connected by $N_b bondswithconstraints$:

```
N_ZM = N_b - d N + N_c
```

where d is spatial dimension, Nc *is number of constraints. When* NZM 0, the structure has **zero modes** (floppy modes).

Topological Polarization: The location of zero modes (bulk vs edge) is determined by topological invariants, analogous to Chern numbers in electronics.

Recent Breakthroughs:

- Kane Lubensky (2014): Topological boundary modes in mechanical systems
- Mechanical Weyl points in 3D metamaterials
- Higher-order topological mechanics with corner modes

0.1.2 Core Question

Can we systematically design mechanical metamaterials with topologically protected zero modes using ONLY graph theory and Maxwell counting—without simulations or trial-and-error?

Specifically:

- Given target number of edge modes $N_edge, constructspring - masslattice$
- Prove zero modes are localized to boundaries using topological invariants
- Optimize stiffness matrix for maximum robustness
- Export 3D-printable designs with exact force constants
- Certify protection against bond disorder and manufacturing defects

0.1.3 Why This Matters

Theoretical Impact:

- Extends topological physics to classical systems
- Connects combinatorics (graph theory) to continuum mechanics
- Provides designer materials with programmable response

Practical Benefits:

- Shock-absorbing materials with directional stiffness
- Soft robotics with controlled compliance
- Mechanical computers (logic gates from topological modes)
- Metamaterial actuators and sensors

Pure Thought Advantages:

- Stiffness matrices are purely combinatorial (connectivity + spring constants)
- Zero modes found via kernel of dynamical matrix
- Topology determined from graph structure alone
- No material properties needed (geometry dominates)
- Directly 3D-printable (with specified spring constants)

0.2 2. Mathematical Formulation

0.2.1 Problem Definition

A **mechanical metamaterial** is a network of N sites (masses) at positions $r_i connected by springs$:
Dynamical Matrix (small oscillations):

```
D_{ij}^{    } =  _ {   ik   } k_{ik} [( _ {ij} -  _ {jk})  _ {    } -
   (r_i - r_k) _   (r_i - r_k) _   / |r_i - r_k|  ]
```

where:

- $k_{ik} is spring constant for bond <ik>$
- , are spatial components (x,y,z)
- ⟨ik⟩ denotes bonded pairs

Zero Modes: Solutions to D = 0 (floppy modes with $^2 = 0$).
Topological Polarization (1D chain):

```
P = (1/2  )   _ {BZ} A(k) dk  mod 1
```

where A(k) is geometric phase (analog of Berry connection for phonons).
When P is fractional, the system has **topological boundary modes**.
Spectral Flow: Number of zero modes crossing from - to + eigenvalues when interpolating between two configurations gives topological index.

0.2.2 Certificate Requirements

- **Maxwell Count Certificate**: Verify NZM = Nb - dN + N$_c$*exactly*

- **Zero Mode Localization**: Prove modes decay exponentially from edges
- **Topological Invariant**: Compute polarization P or winding number
- **Robustness Proof**: Show edge modes survive spring constant disorder k/k
- **Fabrication Specs**: Export spring constants and geometry for 3D printing

0.2.3 Input/Output Specification

Input:

```
from sympy import *
import numpy as np
import networkx as nx
from typing import List, Tuple

class MechanicalLattice:
    dimension: int  # 1D, 2D, or 3D
    sites: List[np.ndarray]  # Position vectors
    bonds: List[Tuple[int, int]]  # Connectivity (i, j) pairs
    spring_constants: dict  # k_{ij} for each bond
    constraints: List[Callable]  # Constraint functions (e.g., fixed
       sites)
```

Output:

```
class MechanicalCertificate:
    lattice: MechanicalLattice

    # Maxwell counting
    N_sites: int
    N_bonds: int
    N_constraints: int
    N_zero_modes_predicted: int

    # Spectrum
    dynamical_matrix: np.ndarray
    eigenvalues: np.ndarray  #      values
    zero_mode_eigenvectors: List[np.ndarray]  # Modes with          0

    # Topology
    topological_polarization: float  # P mod 1
    winding_number: Optional[int]  # For 1D systems
    edge_mode_count: int  # Actual number of boundary modes

    # Localization
    participation_ratio: List[float]  # For each zero mode
    localization_length: float  # Decay length from edge

    # Robustness
    disorder_threshold: float  # Max  k /k before gap closes
    fabrication_tolerance: dict  # Geometric tolerances
```

```
    # Fabrication
    stl_file: Path
    spring_specifications: dict  # Material + geometry for each spring
    assembly_instructions: str
```

0.3 3. Implementation Approach

0.3.1 Phase 1: Maxwell Counting and Zero Modes (Months 1-2)

Implement basic topological mechanics infrastructure:

```
import numpy as np
from scipy.linalg import eigh, null_space
import networkx as nx

def build_dynamical_matrix(lattice: MechanicalLattice) -> np.ndarray:
    """
    Construct dynamical matrix D for small oscillations.

    D_ij^{    } encodes spring network topology and stiffnesses.
    """
    N = len(lattice.sites)
    d = lattice.dimension
    D = np.zeros((N*d, N*d))

    for (i, j), k_ij in zip(lattice.bonds,
       lattice.spring_constants.values()):
        r_ij = lattice.sites[j] - lattice.sites[i]
        dist = np.linalg.norm(r_ij)

        # Spring tensor: k_ij * (r_ij     r_ij) / |r_ij|
        spring_tensor = k_ij * np.outer(r_ij, r_ij) / dist**2

        # Add to dynamical matrix (block structure)
        for    in range(d):
            for    in range(d):
                # Diagonal blocks (i,i) and (j,j)
                D[i*d +   , i*d +   ] += spring_tensor[  ,   ]
                D[j*d +   , j*d +   ] += spring_tensor[  ,   ]

                # Off-diagonal blocks (i,j) and (j,i)
                D[i*d +   , j*d +   ] -= spring_tensor[  ,   ]
                D[j*d +   , i*d +   ] -= spring_tensor[  ,   ]

    return D

def maxwell_count(lattice: MechanicalLattice) -> int:
    """
    Compute predicted number of zero modes from Maxwell counting.

    N_ZM = N_bonds - d*N_sites + N_constraints
    """
    N_sites = len(lattice.sites)
```

```
    N_bonds = len(lattice.bonds)
    d = lattice.dimension
    N_constraints = len(lattice.constraints)

    N_ZM = N_bonds - d*N_sites + N_constraints

    return N_ZM

def find_zero_modes(D: np.ndarray, tolerance: float = 1e-8) ->
   List[np.ndarray]:
    """
    Find zero eigenvalue modes of dynamical matrix.

    Zero modes satisfy D    = 0.
    """
    eigenvalues, eigenvectors = eigh(D)

    # Identify zero modes
    zero_mode_indices = np.where(np.abs(eigenvalues) < tolerance)[0]

    zero_modes = [eigenvectors[:, idx] for idx in zero_mode_indices]

    return zero_modes, eigenvalues[zero_mode_indices]

def remove_trivial_zero_modes(zero_modes: List[np.ndarray],
                              lattice: MechanicalLattice) ->
                                 List[np.ndarray]:
    """
    Remove trivial zero modes (rigid translations/rotations).

    In d dimensions:
    - d translational zero modes
    - d(d-1)/2 rotational zero modes (for free boundary conditions)

    These are not topological.
    """
    N = len(lattice.sites)
    d = lattice.dimension

    # Generate translations
    translations = []
    for    in range(d):
        trans = np.zeros(N*d)
        trans[  ::d] = 1.0  # All sites move in direction
        translations.append(trans)

    # Generate rotations (for d=2: around z-axis; d=3: around x,y,z)
    rotations = []
    if d == 2:
        # Rotation around origin
        rot = np.zeros(N*d)
        for i, r in enumerate(lattice.sites):
            rot[2*i] = -r[1]  #  x  = -y
            rot[2*i+1] = r[0]  #  y  = x
        rotations.append(rot)

```

```
    trivial_modes = translations + rotations

    # Project out trivial modes
    nontrivial_modes = []
    for mode in zero_modes:
        # Check if mode is combination of trivial modes
        projection = sum(np.dot(mode, triv)**2 for triv in
            trivial_modes)

        if projection / np.dot(mode, mode) < 0.99:  # Less than 99%
            overlap
            nontrivial_modes.append(mode)

    return nontrivial_modes
```

Validation: Reproduce Kane-Lubensky kagome lattice with topological zero modes.

0.3.2 Phase 2: Topological Polarization (Months 2-3)

Compute topological invariants for 1D chains:

```
def compute_polarization_1d(lattice: MechanicalLattice, k_values:
    np.ndarray) -> float:
    """
    Compute topological polarization P for 1D mechanical chain.

    P = (1/2  )     A(k) dk  mod 1

    where A(k) is geometric phase (Berry connection for phonons).
    """
    # Build k-space dynamical matrix
    def D_k(k: float) -> np.ndarray:
        # Fourier transform of spring network
        N_unit_cell = count_sites_per_unit_cell(lattice)
        D = np.zeros((N_unit_cell, N_unit_cell), dtype=complex)

        for (i, j) in lattice.bonds:
            # Distance in unit cells
            delta_n = unit_cell_difference(lattice, i, j)

            k_ij = lattice.spring_constants[(i, j)]
            r_ij = lattice.sites[j] - lattice.sites[i]

            phase = np.exp(1j * k * delta_n)
            D[i % N_unit_cell, j % N_unit_cell] += k_ij * phase

        return D

    # Compute Berry connection A(k)
    A_values = []

    for k in k_values:
        D_k_val = D_k(k)
        evals, evecs = eigh(D_k_val)

        # Occupied band (lowest non-zero mode)
```

```
        occupied_state = evecs[:, 0]  # Assuming first mode is occupied

        # Derivative approximation
        dk = k_values[1] - k_values[0]
        D_k_plus = D_k(k + dk)
        evals_plus, evecs_plus = eigh(D_k_plus)
        occupied_plus = evecs_plus[:, 0]

        # Berry connection: A =  i u |  _k  | u
        overlap = np.vdot(occupied_state, occupied_plus)
        A_k = np.imag(np.log(overlap)) / dk

        A_values.append(A_k)

    # Integrate
    P = np.trapz(A_values, k_values) / (2*np.pi)

    return P % 1  # Polarization mod 1

def predict_edge_modes_from_polarization(P: float, N_unit_cells: int)
    -> int:
    """
    Predict number of edge modes from polarization.

    If P     0, there are topological boundary modes.
    Number of modes     P * N_unit_cells (for finite system).
    """
    if abs(P) < 0.1 or abs(P - 1) < 0.1:
        # Trivial polarization
        return 0
    else:
        # Non-trivial: edge modes present
        # Exact count depends on termination
        return int(np.round(abs(P)))
```

0.3.3 Phase 3: 2D Kagome Lattice (Months 3-4)

Implement canonical topological mechanical system:

```
def kagome_lattice(N_x: int, N_y: int, spring_const: float = 1.0) ->
    MechanicalLattice:
    """
    Construct kagome lattice with topological edge modes.

    Kagome lattice: 3 sites per unit cell, triangular Bravais lattice.
    Has Z = 1 (one topological edge mode per edge).
    """
    sites = []
    bonds = []

    # Lattice vectors
    a1 = np.array([1.0, 0.0])
    a2 = np.array([0.5, np.sqrt(3)/2])

    # Three sublattice positions within unit cell
    r_A = np.array([0, 0])
```

```
    r_B = a1 / 2
    r_C = a2 / 2

    site_index = {}

    for nx in range(N_x):
        for ny in range(N_y):
            origin = nx*a1 + ny*a2

            # Add 3 sites
            idx_A = len(sites)
            idx_B = len(sites) + 1
            idx_C = len(sites) + 2

            sites.append(origin + r_A)
            sites.append(origin + r_B)
            sites.append(origin + r_C)

            site_index[(nx, ny, 'A')] = idx_A
            site_index[(nx, ny, 'B')] = idx_B
            site_index[(nx, ny, 'C')] = idx_C

            # Intra-cell bonds
            bonds.append((idx_A, idx_B))
            bonds.append((idx_B, idx_C))
            bonds.append((idx_C, idx_A))

    # Inter-cell bonds (with periodic boundary conditions in bulk)
    for nx in range(N_x):
        for ny in range(N_y):
            # Connect to neighboring cells
            idx_C = site_index[(nx, ny, 'C')]

            # Bond to A in (nx+1, ny)
            if nx + 1 < N_x:
                idx_A_right = site_index[(nx+1, ny, 'A')]
                bonds.append((idx_C, idx_A_right))

            # Bond to B in (nx, ny+1)
            if ny + 1 < N_y:
                idx_B_up = site_index[(nx, ny+1, 'B')]
                bonds.append((idx_C, idx_B_up))

    # All springs have same constant
    spring_constants = {bond: spring_const for bond in bonds}

    return MechanicalLattice(
        dimension=2,
        sites=sites,
        bonds=bonds,
        spring_constants=spring_constants,
        constraints=[]
    )

def compute_edge_mode_localization(lattice: MechanicalLattice,
                                   zero_mode: np.ndarray) -> float:
```

```
    """
    Compute localization length of edge mode.

    Fit |  (x)| ~ exp(-x/  ) where    is localization length.
    """
    N = len(lattice.sites)
    d = lattice.dimension

    # Extract displacement amplitudes
    amplitudes = np.array([np.linalg.norm(zero_mode[i*d:(i+1)*d])
                           for i in range(N)])

    # Find edge (assume edge is at x=0)
    x_positions = np.array([site[0] for site in lattice.sites])

    # Sort by distance from edge
    sorted_indices = np.argsort(x_positions)

    # Fit exponential decay
    from scipy.optimize import curve_fit

    def exp_decay(x, xi, A):
        return A * np.exp(-x / xi)

    try:
        popt, _ = curve_fit(exp_decay,
                            x_positions[sorted_indices],
                            amplitudes[sorted_indices],
                            p0=[1.0, 1.0])
        localization_length = popt[0]
    except:
        localization_length = np.inf  # Delocalized

    return localization_length
```

0.3.4 Phase 4: Disorder and Robustness (Months 4-5)

Test topological protection:

```
def add_spring_disorder(lattice: MechanicalLattice,
                        disorder_strength: float) -> MechanicalLattice:
    """
    Add disorder to spring constants: k     k(1 +   ) where    ~ U(-   ,
         ).
    """
    disordered_springs = {}

    for bond, k_0 in lattice.spring_constants.items():
        delta = disorder_strength * np.random.uniform(-1, 1)
        disordered_springs[bond] = k_0 * (1 + delta)

    return MechanicalLattice(
        dimension=lattice.dimension,
        sites=lattice.sites,
        bonds=lattice.bonds,
        spring_constants=disordered_springs,
```

```
        constraints=lattice.constraints
    )

def test_topological_robustness(lattice: MechanicalLattice,
                                disorder_levels: List[float],
                                N_trials: int = 50) -> dict:
    """
    Test edge mode survival vs spring constant disorder.
    """
    results = {}

    # Count edge modes in clean system
    D_clean = build_dynamical_matrix(lattice)
    zero_modes_clean, _ = find_zero_modes(D_clean)
    N_edge_clean = len(remove_trivial_zero_modes(zero_modes_clean,
        lattice))

    for disorder in disorder_levels:
        edge_mode_survival = []

        for trial in range(N_trials):
            disordered = add_spring_disorder(lattice, disorder)
            D_disorder = build_dynamical_matrix(disordered)
            zero_modes, _ = find_zero_modes(D_disorder, tolerance=1e-6)
            N_edge_disorder = len(remove_trivial_zero_modes(zero_modes,
                disordered))

            # Check if edge modes survive
            survival = (N_edge_disorder == N_edge_clean)
            edge_mode_survival.append(survival)

        results[disorder] = {
            'survival_rate': np.mean(edge_mode_survival),
            'mean_edge_modes': np.mean([N_edge_disorder for _ in
                edge_mode_survival])
        }

    return results
```

0.3.5 Phase 5: 3D Printing Export (Months 5-6)

Generate fabrication files:

```
def export_metamaterial_stl(lattice: MechanicalLattice,
                            output_path: Path,
                            rod_radius: float = 0.05,
                            node_radius: float = 0.1):
    """
    Export mechanical metamaterial as STL for 3D printing.

    Each bond becomes a cylindrical rod, each site a spherical node.
    Spring constant encoded in rod cross-section.
    """
    from stl import mesh
    import trimesh

```

```
    geometries = []

    # Create nodes (spheres at sites)
    for site in lattice.sites:
        sphere = trimesh.creation.icosphere(radius=node_radius)
        sphere.apply_translation(site)
        geometries.append(sphere)

    # Create bonds (cylinders)
    for (i, j) in lattice.bonds:
        r_i = lattice.sites[i]
        r_j = lattice.sites[j]

        # Spring constant determines rod thickness
        k_ij = lattice.spring_constants[(i, j)]
        rod_thickness = rod_radius * np.sqrt(k_ij)  # Encode stiffness in geometry

        cylinder = create_cylinder(r_i, r_j, rod_thickness)
        geometries.append(cylinder)

    # Combine all geometries
    combined_mesh = trimesh.util.concatenate(geometries)

    # Export
    combined_mesh.export(str(output_path))

def generate_assembly_instructions(lattice: MechanicalLattice) -> str:
    """
    Generate human-readable assembly protocol.
    """
    instructions = "Mechanical Metamaterial Assembly\n"
    instructions += "=" * 50 + "\n\n"

    instructions += "1. Material: TPU (Shore 95A hardness)\n"
    instructions += "2. Print Settings: 0.1mm layer height, 100% infill\n\n"

    instructions += "3. Spring Constants (encoded in rod diameter):\n"
    for (i, j), k in lattice.spring_constants.items():
        d = 2 * rod_radius * np.sqrt(k)
        instructions += f"   Bond ({i},{j}): k={k:.2f} N/m     diameter {d:.3f} mm\n"

    instructions += "\n4. Expected Zero Modes:\n"
    N_ZM = maxwell_count(lattice)
    instructions += f"   Total: {N_ZM} zero modes\n"
    instructions += f"   Edge modes: {N_ZM - lattice.dimension*(lattice.dimension-1)/2}\n"

    return instructions

def certificate_export(cert: MechanicalCertificate, output_dir: Path):
    """
    Export complete certificate as JSON + STL + instructions.
    """
```

```
    import json

    # JSON certificate
    cert_dict = {
        'maxwell_count': {
            'N_sites': cert.N_sites,
            'N_bonds': cert.N_bonds,
            'N_constraints': cert.N_constraints,
            'N_zero_modes_predicted': cert.N_zero_modes_predicted
        },
        'topology': {
            'polarization': float(cert.topological_polarization),
            'edge_mode_count': cert.edge_mode_count
        },
        'localization': {
            'localization_length': float(cert.localization_length),
            'participation_ratios': [float(pr) for pr in
               cert.participation_ratio]
        },
        'robustness': {
            'disorder_threshold': float(cert.disorder_threshold)
        },
        'fabrication': {
            'stl_file': str(cert.stl_file),
            'tolerances': cert.fabrication_tolerance
        }
    }

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

    # Assembly instructions
    with open(output_dir / 'assembly.txt', 'w') as f:
        f.write(cert.assembly_instructions)
```

0.3.6 Phase 6: Database Generation (Months 6-7)

Catalog topological mechanical systems:

```
def generate_metamaterial_database() -> dict:
    """
    Generate database of topological mechanical metamaterials.
    """
    database = {'models': []}

    # 1D chains
    for topology in ['SSH', 'dimerized']:
        lattice = create_1d_chain(topology, N_cells=20)
        cert = generate_mechanical_certificate(lattice)
        database['models'].append({
            'name': f'1D_{topology}',
            'dimension': 1,
            'edge_modes': cert.edge_mode_count,
            'polarization': cert.topological_polarization
        })

```

```
    # 2D lattices
    for geometry in ['kagome', 'snub_square', 'honeycomb']:
        lattice = create_2d_lattice(geometry, N_x=10, N_y=10)
        cert = generate_mechanical_certificate(lattice)
        database['models'].append({
            'name': f'2D_{geometry}',
            'dimension': 2,
            'edge_modes': cert.edge_mode_count,
            'certificate_path': export_certificate(cert)
        })

    return database
```

0.4 4. Example Starting Prompt

```
You are a mechanical engineer specializing in topological
   metamaterials. Design spring-mass
networks with topologically protected zero modes using ONLY graph
   theory and Maxwell counting.

OBJECTIVE: Construct kagome lattice with Z=1 edge modes, prove
   topological protection,
export 3D-printable STL file.

PHASE 1 (Months 1-2): Maxwell counting infrastructure
- Implement dynamical matrix construction from spring network
- Code zero mode finder (kernel of D)
- Validate on simple chain: SSH model with edge modes
- Verify Maxwell count: N_ZM = N_b - 2N

PHASE 2 (Months 2-3): Topological polarization
- Compute P =   A  (k)dk/(2  ) for 1D chains
- Verify P = 1/2 for SSH model     edge modes
- Implement winding number calculation

PHASE 3 (Months 3-4): Kagome lattice
- Construct 10 10  kagome with N_x, N_y unit cells
- Compute all zero modes via eigh(D)
- Remove trivial modes (translations + rotations)
- Verify edge mode count = perimeter length

PHASE 4 (Months 4-5): Robustness testing
- Add spring disorder: k     k(1    ) with    = 5%, 10%, 20%
- Test edge mode survival (50 realizations per disorder level)
- Find critical disorder  _c  where modes disappear

PHASE 5 (Months 5-6): Fabrication export
- Generate STL with spherical nodes, cylindrical bonds
- Encode spring constants in rod thickness: d =  d _ 0 k
- Specify material: TPU flexible filament
- Write assembly instructions

PHASE 6 (Months 6-7): Database generation
```

```
- Catalog 5 different 2D topological lattices
- Export certificates for each
- Compare edge mode counts, localization lengths

SUCCESS CRITERIA:
- MVR: SSH and kagome lattices with verified zero modes
- Strong: Topological polarization computed, disorder tested
- Publication: Database of 10 metamaterials with STL files

VERIFICATION:
- Maxwell count exact: N_ZM = N_b - dN (integer)
- Zero modes verified: |  | < 10
- Edge localization:    < 3 lattice spacings
- Disorder threshold:  _c  > 20%

Pure combinatorics + linear algebra. No FEM simulations.
All results certificate-based with exact arithmetic.
```

0.5 5. Success Criteria

0.5.1 MVR (2 months)

- SSH chain + kagome with verified zero modes
- Maxwell counting validated

0.5.2 Strong (4-5 months)

- Polarization computed for 5 systems
- Disorder robustness tested
- 10 metamaterials cataloged

0.5.3 Publication (6-7 months)

- Complete database with STL files
- Formal proofs of edge mode counts
- Application to soft robotics

0.6 6. Verification Protocol

Automated checks: Maxwell count, zero mode kernel, edge localization fits, disorder statistics.

0.7 7. Resources Milestones

References:

- Kane Lubensky (2014): "Topological Boundary Modes in Isostatic Lattices"
- Paulose et al. (2015): "Topological Modes Bound to Dislocations"
- Nash et al. (2015): "Topological Mechanics of Gyroscopic Metamaterials"

Milestones:

- Month 2: SSH + kagome validated
- Month 4: Polarization working
- Month 6: Database + STL export complete

0.8 8. Extensions

- **3D Weyl Mechanics**: Mechanical Weyl points
- **Active Metamaterials**: Motors + topological modes
- **Quantum Mechanics**: Phonon topology in quantum crystals

End of PRD 12