# PRD 11: Photonic Topological Crystals from Symmetry

Pure Thought AI Challenge 11

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**: 4-6 months
**Difficulty**: Medium-High
**Prerequisites**: Electromagnetism, group theory, computational linear algebra, photonic band theory

---

## 0.1 1. Problem Statement

### 0.1.1 Scientific Context

**Photonic crystals** are periodic dielectric structures that create photonic band gaps—frequency ranges where light cannot propagate. When combined with **topological band theory**, they enable:

- **Topologically Protected Edge States**: Light propagates along interfaces without backscattering
- **Robust Waveguides**: Immune to disorder, sharp bends, defects
- **Unidirectional Propagation**: Optical isolators, non-reciprocal devices
- **Topological Lasers**: Single-mode operation enforced by topology

The key advantage of photonic systems is **complete theoretical predictability**:

- Maxwell's equations are exactly solvable for periodic structures
- No quantum many-body effects to worry about
- Band structures computed from pure geometry + refractive indices
- Fabrication-ready designs (3D printing, lithography)

**Recent Developments**:

- Topological photonic crystals realized in microwave, optical, THz regimes
- Chern insulator analogs using gyromagnetic materials (breaking time-reversal)
- topological photonics using bianisotropic metamaterials
- Higher-order topological photonics with corner states

### 0.1.2 Core Question

**Can we design photonic crystal structures with non-trivial topology using ONLY symmetry principles and Maxwell's equations—without any experimental input or trial-and-error?**

Specifically:

- Given target photonic Chern number C, construct periodic dielectric arrangement
- Prove existence of topologically protected edge modes
- Optimize geometry for largest photonic band gap
- Certify robustness against realistic fabrication imperfections
- Export fabrication-ready blueprints (STL files for 3D printing)

### 0.1.3 Why This Matters

**Theoretical Impact**:

- Demonstrates pure-thought materials design from first principles
- Connects abstract topology to electromagnetic engineering
- Validates certificate-based approach to metamaterial discovery

**Practical Benefits**:

- Produces directly fabricable designs for optical devices
- Enables robust optical communication (disorder-immune waveguides)
- Applications: optical isolators, topological lasers, quantum photonics

**Pure Thought Advantages**:

- Maxwell's equations are exact (no approximations needed)
- Band structures computed via eigenvalue problems
- Symmetry analysis is purely group-theoretic
- No experimental measurements required

---

## 0.2 2. Mathematical Formulation

### 0.2.1 Problem Definition

A **photonic crystal** is a periodic arrangement of dielectric materials with permittivity (r) = (r + R) for lattice vectors R.

**Master Equation** (frequency-domain Maxwell):

```
1  (        E(r)) = (  /c)      (r) E(r)
```

Using Bloch's theorem: E(r) = e$^{(}$*ik*ů*r*)*uk(r) where uk*$(r + R) = u_k(r)$.

This becomes an eigenvalue problem:

```
1  _k   u_k = ( _k /c)   u_k
```

where $_k = (r)$ź[+*ik*]Ö[+*ik*]Ö*isthemasteroperator*.

**Photonic Band Structure**: Eigenvalues $_n(k)$*arephotonicbands*(*analogoustoelectronicbands*).

**Topological Invariants**:

For photonic Chern insulators (time-reversal broken by gyromagnetic materials):

```
1  C = (1/2 i )   _  {BZ} Tr[P (  _  {k_x} P   _  {k_y} P -   _  {k_y} P
       _  {k_x} P)] dk
```

where P(k) = $n$ /u$_n$(k)⟩⟨u$_n$(*k*)|*projectsontofilledbands*.

For topological photonics (time-reversal preserved):

```
1  =  _ {i     TRIM}  _i   mod 2
```

where ¡*areparityeigenvaluesattime* − *reversalinvariantmomenta*.

### 0.2.2 Certificate Requirements

Given a photonic crystal design:

- **Band Gap Certificate**: Prove  frequency range [, ] with no propagating modes
- **Chern Number Certificate**: Compute exact C via Berry curvature integration
- **Edge State Certificate**: Demonstrate localized modes at interface
- **Robustness Certificate**: Prove edge states survive disorder in (r)
- **Fabrication Blueprint**: Export geometry as STL/CAD file with tolerances

### 0.2.3 Input/Output Specification

**Input**:

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

class PhotonicCrystal:
    dimension: int  # 2D or 3D
    lattice_vectors: List[np.ndarray]  # Bravais lattice
    permittivity_func: Callable[[np.ndarray], complex]  #   (r)
    permeability_func: Callable[[np.ndarray], complex]  #   (r)
       (usually =1)

    # For topological designs
    target_chern: Optional[int]
    target_gap_width: float  # Desired     /
```

**Output**:

```python
class PhotonicCertificate:
    crystal: PhotonicCrystal

    # Band structure
    band_structure: np.ndarray  #  _n (k) for all bands n, momenta k
    band_gap: Tuple[float, float]  # (  _lower  ,  _upper  )
    gap_to_midgap_ratio: float  #      /   _midgap

    # Topology
    chern_number: int
    berry_curvature: Callable[[np.ndarray], float]  # F(k)
    z2_invariant: Optional[int]  # For TR-invariant systems

    # Edge states
    edge_dispersion: np.ndarray  #   (k_parallel) for ribbon geometry
    localization_length: float  # Penetration depth into bulk

    # Robustness
    disorder_threshold: float  # Max      before gap closes
    fabrication_tolerance: float  # Max geometric error

    # Fabrication
```

```
    stl_file: Path  # 3D printable geometry
    refractive_index_profile: np.ndarray  # For lithography

    # Verification
    simulation_log: str  # FDTD or planewave expansion results
    proof_of_topology: str  # Mathematical derivation
```

---

## 0.3 3. Implementation Approach

### 0.3.1 Phase 1: Plane Wave Expansion Method (Month 1)

Implement standard photonic band structure solver:

```
import numpy as np
from scipy.linalg import eigh
from scipy.sparse.linalg import eigsh
import itertools

def generate_reciprocal_lattice(real_lattice: List[np.ndarray], N_max:
    int = 5) -> List[np.ndarray]:
    """
    Generate reciprocal lattice vectors G for plane wave expansion.

    For 2D square lattice: G = 2  ( n  ,  n  )/a for | n  |, | n  |
       N_max
    """
    # Compute reciprocal lattice basis
    if len(real_lattice) == 2:
        a1, a2 = real_lattice
        b1 = 2*np.pi * np.array([a2[1], -a2[0]]) / (a1[0]*a2[1] -
           a1[1]*a2[0])
        b2 = 2*np.pi * np.array([-a1[1], a1[0]]) / (a1[0]*a2[1] -
           a1[1]*a2[0])

        G_vectors = []
        for n1 in range(-N_max, N_max+1):
            for n2 in range(-N_max, N_max+1):
                G_vectors.append(n1*b1 + n2*b2)

    return G_vectors

def fourier_coefficients_permittivity(eps_func: Callable, lattice:
    List[np.ndarray],
                                      G_vectors: List[np.ndarray]) ->
                                         dict:
    """
    Compute Fourier coefficients  _G  of permittivity.

      (r) =  _G   _G  e^{iG r}

    Uses FFT on fine real-space grid.
    """
    # Real-space grid
    N_grid = 128
```

```
    x_grid = np.linspace(0, np.linalg.norm(lattice[0]), N_grid)
    y_grid = np.linspace(0, np.linalg.norm(lattice[1]), N_grid)

    eps_real = np.zeros((N_grid, N_grid), dtype=complex)

    for i, x in enumerate(x_grid):
        for j, y in enumerate(y_grid):
            r = np.array([x, y])
            eps_real[i, j] = eps_func(r)

    # FFT
    eps_fourier = np.fft.fft2(eps_real) / (N_grid**2)

    # Extract coefficients for G_vectors
    eps_G = {}
    for G in G_vectors:
        # Map G to FFT index
        idx = reciprocal_to_fft_index(G, lattice, N_grid)
        eps_G[tuple(G)] = eps_fourier[idx[0], idx[1]]

    return eps_G

def build_master_operator(k: np.ndarray, G_vectors: List[np.ndarray],
                          eps_G: dict) -> np.ndarray:
    """
    Construct master operator   _k   in plane wave basis.

    For TE modes (E_z only in 2D):
    [  _k  ]_{G,G'} = (k+G)    (k+G')  _ {G,G'}     _G     - (k+G)
       (k+G')  _ {G-G'}

    (Simplified for 2D)
    """
    N_G = len(G_vectors)
    Theta = np.zeros((N_G, N_G), dtype=complex)

    for i, G in enumerate(G_vectors):
        for j, G_prime in enumerate(G_vectors):
            k_plus_G = k + G
            k_plus_G_prime = k + G_prime

            if i == j:
                # Diagonal term
                Theta[i, j] = np.dot(k_plus_G, k_plus_G) /
                    eps_G[tuple(np.zeros(2))]
            else:
                # Off-diagonal
                G_diff = tuple(G - G_prime)
                if G_diff in eps_G:
                    Theta[i, j] = -np.dot(k_plus_G, k_plus_G_prime) /
                        eps_G[G_diff]

    return Theta

def compute_photonic_bands(crystal: PhotonicCrystal, k_path:
    List[np.ndarray],
```

```
                              N_bands: int = 10) -> np.ndarray:
    """
    Compute photonic band structure  _n (k) along k_path.

    Returns array of shape (len(k_path), N_bands) with frequencies   /c.
    """
    G_vectors = generate_reciprocal_lattice(crystal.lattice_vectors,
       N_max=5)
    eps_G = fourier_coefficients_permittivity(crystal.permittivity_func,
                                              crystal.lattice_vectors,
                                              G_vectors)

    bands = np.zeros((len(k_path), N_bands))

    for i, k in enumerate(k_path):
        Theta = build_master_operator(k, G_vectors, eps_G)

        # Solve eigenvalue problem:    u = (  /c)   u
        eigenvalues, eigenvectors = eigh(Theta)

        #    = c       (take positive root)
        frequencies = np.sqrt(np.abs(eigenvalues[:N_bands]))
        bands[i, :] = frequencies

    return bands

def identify_band_gap(bands: np.ndarray) -> Tuple[float, float]:
    """
    Find largest photonic band gap.

    Returns ( _lower  ,  _upper  ) in units of c/a.
    """
    N_k, N_bands = bands.shape

    gaps = []

    for n in range(N_bands - 1):
        # Gap between band n and n+1
        upper_edge_n = np.max(bands[:, n])
        lower_edge_n1 = np.min(bands[:, n+1])

        if lower_edge_n1 > upper_edge_n:
            gap_size = lower_edge_n1 - upper_edge_n
            gaps.append((upper_edge_n, lower_edge_n1, gap_size))

    if gaps:
        # Return largest gap
        largest_gap = max(gaps, key=lambda x: x[2])
        return (largest_gap[0], largest_gap[1])
    else:
        return (0, 0)  # No gap
```

**Validation**: Reproduce known band structure for square lattice of dielectric rods.

### 0.3.2 Phase 2: Topological Design via Symmetry Breaking (Months 2-3)

Design photonic Chern insulators by breaking time-reversal symmetry:

```
def gyromagnetic_photonic_crystal(lattice_type: str = 'honeycomb',
                                  gyromagnetic_strength: float = 0.1)
                                     -> PhotonicCrystal:
    """
    Construct photonic Chern insulator using gyromagnetic materials.

    Gyromagnetic materials:    is a tensor with off-diagonal elements
    (breaks time-reversal symmetry, like magnetic field for electrons).

       = [[     , i    , 0],
         [-i    ,      , 0],
         [0, 0,  _z ]]

    where    is gyromagnetic coupling (proportional to B-field).
    """
    if lattice_type == 'honeycomb':
        # Honeycomb lattice (analogous to graphene for photons)
        a1 = np.array([1, 0])
        a2 = np.array([0.5, np.sqrt(3)/2])
        lattice_vectors = [a1, a2]

        def eps_honeycomb(r: np.ndarray) -> complex:
            # Two sublattices with gyromagnetic rods
            rod_radius = 0.2

            # Sublattice A at origin
            if np.linalg.norm(r) < rod_radius:
                # Gyromagnetic permittivity (complex tensor
                    effective scalar)
                return 12.0 * (1 + 1j*gyromagnetic_strength)

            # Sublattice B at (a1 + a2)/3
            r_B = r - (a1 + a2) / 3
            if np.linalg.norm(r_B) < rod_radius:
                return 12.0 * (1 - 1j*gyromagnetic_strength)  #
                    Opposite sign

            # Background
            return 1.0

    else:
        raise ValueError(f"Unknown lattice type: {lattice_type}")

    return PhotonicCrystal(
        dimension=2,
        lattice_vectors=lattice_vectors,
        permittivity_func=eps_honeycomb,
        permeability_func=lambda r: 1.0,
        target_chern=1,
        target_gap_width=0.1
    )

def optimize_for_band_gap(initial_crystal: PhotonicCrystal,
```

```
                            param_ranges: dict) -> PhotonicCrystal:
    """
    Optimize crystal parameters to maximize photonic band gap.

    Parameters: rod radius, permittivity contrast, gyromagnetic
       strength, etc.
    """
    from scipy.optimize import minimize

    def objective(params):
        # Update crystal with new parameters
        crystal = update_crystal_parameters(initial_crystal, params)

        # Compute band structure
        k_path = generate_k_path(crystal.lattice_vectors, N_k=50)
        bands = compute_photonic_bands(crystal, k_path)

        # Find gap
        gap_lower, gap_upper = identify_band_gap(bands)
        gap_size = gap_upper - gap_lower

        # Maximize gap (negative because we minimize)
        return -gap_size

    # Constraints: maintain topology
    constraints = [
        {'type': 'eq', 'fun': lambda p: verify_chern_preserved(p,
           initial_crystal.target_chern)}
    ]

    result = minimize(objective, x0=list(param_ranges.values()),
                      method='SLSQP', constraints=constraints)

    return update_crystal_parameters(initial_crystal, result.x)
```

### 0.3.3 Phase 3: Berry Curvature and Chern Number (Months 3-4)

Compute topological invariants for photonic bands:

```
def photonic_berry_connection(crystal: PhotonicCrystal, band_indices:
   List[int],
                              k: np.ndarray, delta: float = 1e-5) ->
                                 np.ndarray:
    """
    Compute Berry connection A_ (k) for photonic bands.

    Same formula as electronic case, but wavefunctions are now
    electromagnetic field patterns u_n(k).
    """
    G_vectors = generate_reciprocal_lattice(crystal.lattice_vectors)
    eps_G = fourier_coefficients_permittivity(crystal.permittivity_func,
                                              crystal.lattice_vectors,
                                              G_vectors)

    # Get eigenvectors at k
    Theta_k = build_master_operator(k, G_vectors, eps_G)
```

```
    evals, evecs = eigh(Theta_k)

    # Select occupied bands
    occupied_states = evecs[:, band_indices]

    A = np.zeros(2, dtype=complex)

    for mu in range(2):
        dk = np.zeros(2)
        dk[mu] = delta

        # Eigenvectors at k + dk
        Theta_k_plus = build_master_operator(k + dk, G_vectors, eps_G)
        evals_plus, evecs_plus = eigh(Theta_k_plus)
        occupied_plus = evecs_plus[:, band_indices]

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

    return A

def compute_photonic_chern(crystal: PhotonicCrystal, band_indices:
    List[int],
                           N_k: int = 100) -> int:
    """
    Compute Chern number for photonic bands via Berry curvature
        integration.
    """
    # Discretize Brillouin zone
    b1, b2 = compute_reciprocal_basis(crystal.lattice_vectors)

    kx_grid = np.linspace(0, 1, N_k, endpoint=False)
    ky_grid = np.linspace(0, 1, N_k, endpoint=False)

    chern_integral = 0.0

    for kx_frac in kx_grid:
        for ky_frac in ky_grid:
            k = kx_frac*b1 + ky_frac*b2

            # Berry curvature
            F = photonic_berry_curvature(crystal, band_indices, k)
            chern_integral += F

    # Normalize
    chern_integral *= (1.0 / N_k)**2 / (2*np.pi)

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

### 0.3.4 Phase 4: Edge State Calculation (Months 4-5)

Compute topologically protected edge modes:

```
def photonic_ribbon_geometry(crystal: PhotonicCrystal, width: int = 50)
    -> Callable:
```

```
    """
    Construct ribbon with open boundary in one direction for edge state
       calculation.

    Similar to electronic case but using photonic master operator.
    """
    def eps_ribbon(r: np.ndarray) -> complex:
        # Check if r is within ribbon bounds
        if 0 <= r[1] < width *
           np.linalg.norm(crystal.lattice_vectors[1]):
            return crystal.permittivity_func(r)
        else:
            return 1.0  # Vacuum outside

    return eps_ribbon

def compute_edge_states_photonic(crystal: PhotonicCrystal, width: int =
   50) -> np.ndarray:
    """
    Compute photonic edge state dispersion   (k_x) for ribbon geometry.
    """
    ribbon_eps = photonic_ribbon_geometry(crystal, width)

    # Modify crystal to ribbon
    crystal_ribbon = PhotonicCrystal(
        dimension=2,
        lattice_vectors=crystal.lattice_vectors,
        permittivity_func=ribbon_eps,
        permeability_func=crystal.permeability_func
    )

    # Compute band structure along k_x
    k_parallel_values = np.linspace(0,
       2*np.pi/np.linalg.norm(crystal.lattice_vectors[0]), 200)
    edge_spectrum = []

    for k_par in k_parallel_values:
        k = np.array([k_par, 0])
        bands_ribbon = compute_photonic_bands(crystal_ribbon, [k],
           N_bands=50)
        edge_spectrum.append(bands_ribbon[0, :])

    return np.array(edge_spectrum)

def visualize_edge_mode(crystal: PhotonicCrystal, k_parallel: float,
                        freq: float, width: int = 50) -> np.ndarray:
    """
    Compute electromagnetic field pattern of edge mode.

    Returns: E_z(x, y) field distribution
    """
    # Solve for eigenmode at (k_parallel, freq)
    # ... (FDTD or eigenmode solver implementation)

    E_field = solve_edge_eigenmode(crystal, k_parallel, freq, width)

```

```
    return E_field
```

### 0.3.5 Phase 5: Robustness Certification (Months 5-6)

Verify edge states survive realistic disorder:

```
def add_disorder_to_crystal(crystal: PhotonicCrystal,
                            disorder_strength: float) -> PhotonicCrystal:
    """
    Add random disorder to permittivity:           +      where      ~
       U(-   , +   ).
    """
    def eps_disordered(r: np.ndarray) -> complex:
        eps_clean = crystal.permittivity_func(r)

        # Random perturbation (spatially smooth)
        delta_eps = disorder_strength * np.random.uniform(-1, 1)

        return eps_clean + delta_eps

    return PhotonicCrystal(
        dimension=crystal.dimension,
        lattice_vectors=crystal.lattice_vectors,
        permittivity_func=eps_disordered,
        permeability_func=crystal.permeability_func,
        target_chern=crystal.target_chern,
        target_gap_width=crystal.target_gap_width
    )

def test_topological_protection(crystal: PhotonicCrystal,
                                disorder_levels: List[float],
                                N_realizations: int = 50) -> dict:
    """
    Test edge state robustness against disorder.

    Returns: statistics on edge state survival vs disorder strength.
    """
    results = {}

    for disorder in disorder_levels:
        edge_state_count = []

        for trial in range(N_realizations):
            disordered_crystal = add_disorder_to_crystal(crystal,
               disorder)
            edge_spectrum =
               compute_edge_states_photonic(disordered_crystal)

            # Count edge states in gap
            gap_lower, gap_upper = identify_band_gap(...)
            edge_count = count_states_in_gap(edge_spectrum, gap_lower,
               gap_upper)

            edge_state_count.append(edge_count)

        results[disorder] = {
```

```
            'mean_edge_count': np.mean(edge_state_count),
            'std_edge_count': np.std(edge_state_count),
            'survival_probability': np.mean(np.array(edge_state_count)
                > 0)
        }

    return results
```

### 0.3.6 Phase 6: Fabrication Export (Month 6)

Generate CAD files for 3D printing / lithography:

```
def export_to_stl(crystal: PhotonicCrystal, output_path: Path,
                  N_unit_cells: Tuple[int, int, int] = (5, 5, 1),
                  resolution: int = 100):
    """
    Export photonic crystal geometry as STL file for 3D printing.

    High    regions become solid material, low    regions are air.
    """
    from stl import mesh

    # Generate 3D voxel grid
    Nx, Ny, Nz = [N * resolution for N in N_unit_cells]

    x = np.linspace(0, N_unit_cells[0] * crystal.lattice_vectors[0][0],
        Nx)
    y = np.linspace(0, N_unit_cells[1] * crystal.lattice_vectors[1][1],
        Ny)
    z = np.linspace(0, 1, Nz)  # Thickness in z

    voxel_grid = np.zeros((Nx, Ny, Nz), dtype=bool)

    for i, xi in enumerate(x):
        for j, yj in enumerate(y):
            for k, zk in enumerate(z):
                r = np.array([xi, yj, zk])
                eps_val = np.real(crystal.permittivity_func(r[:2]))

                # Threshold: high    = solid
                voxel_grid[i, j, k] = (eps_val > 5.0)

    # Convert voxels to mesh (marching cubes)
    vertices, faces = voxels_to_mesh(voxel_grid, x, y, z)

    # Create STL mesh
    crystal_mesh = mesh.Mesh(np.zeros(faces.shape[0],
        dtype=mesh.Mesh.dtype))
    for i, face in enumerate(faces):
        for j in range(3):
            crystal_mesh.vectors[i][j] = vertices[face[j]]

    crystal_mesh.save(str(output_path))

def generate_fabrication_instructions(crystal: PhotonicCrystal) -> str:
    """
```

```
    Generate human-readable fabrication protocol.
    """
    instructions = "Photonic Crystal Fabrication Instructions\n"
    instructions += "=" * 50 + "\n\n"

    instructions += "1. Material Selection:\n"
    instructions += f"   - High-   regions: n =
       {np.sqrt(max_permittivity):.2f}\n"
    instructions += "   - Background: Air (n = 1.0)\n\n"

    instructions += "2. Geometry:\n"
    instructions += f"   - Lattice: {crystal.lattice_vectors}\n"
    instructions += f"   - Unit cell size:
       {np.linalg.norm(crystal.lattice_vectors[0]):.3f}  m \n\n"

    instructions += "3. Fabrication Method:\n"
    instructions += "   - 3D Printing: Use STL file (resolution 10
        m )\n"
    instructions += "   - Lithography: Multi-layer stack (see
       layer-by-layer specs)\n\n"

    instructions += "4. Tolerances:\n"
    instructions += f"   - Position accuracy:
         {fabrication_tolerance:.2f}  m \n"
    instructions += f"   - Refractive index:   {index_tolerance:.3f}\n"

    return instructions
```

---

## 0.4 4. Example Starting Prompt

```
You are a photonics engineer specializing in topological metamaterials.
   Your task is to design
photonic crystals with non-trivial topology using ONLY Maxwell's
   equations and symmetryno
experimental data or trial-and-error allowed.

OBJECTIVE: Construct 2D photonic Chern insulator with C = 1, prove
   topological protection,
and export fabrication-ready STL files.

PHASE 1 (Month 1): Plane wave expansion solver
- Implement Fourier expansion of permittivity   (r)
- Construct master operator   _k   in plane wave basis
- Compute photonic band structure for square lattice test case
- Validate against known results (dielectric rods in air)

PHASE 2 (Months 2-3): Topological design
- Implement gyromagnetic photonic crystal on honeycomb lattice
- Add time-reversal breaking (       0 in permittivity tensor)
- Optimize rod radius and   -contrast for maximum band gap
- Target:     /      > 10%

PHASE 3 (Months 3-4): Topology calculation
```

```
- Compute Berry curvature F(k) for photonic bands
- Integrate to find Chern number C
- Verify C =  1  using Fukui lattice gauge method
- Check uniformity of F(k) across Brillouin zone

PHASE 4 (Months 4-5): Edge states
- Construct ribbon geometry (open boundary in y-direction)
- Solve for edge modes   (k_x) in photonic band gap
- Visualize electromagnetic field patterns E(x,y)
- Verify unidirectional propagation (group velocity has fixed sign)

PHASE 5 (Month 5): Robustness testing
- Add random disorder to   (r):     /   ~ 5%, 10%, 20%
- Compute edge state survival vs disorder strength
- Certify topological protection: edge states persist until      >
      _critical

PHASE 6 (Month 6): Fabrication export
- Generate STL file for 3D printing (5 5  unit cells)
- Specify materials:  T i O  (n=2.4) rods in air
- Write fabrication protocol with tolerances
- Predict operating frequency: ~10 THz (mid-infrared)

SUCCESS CRITERIA:
- MVR: Band structure solver working, gap identified for test structure
- Strong: Photonic Chern insulator with C=1, edge states computed
- Publication: Fabrication-ready design, robustness certified, STL
    exported

VERIFICATION:
- Band gap verified:  _gap  /  _mid  > 10%
- Chern number exact: C = 1 (Fukui method, integer result)
- Edge states localized: decay length < 3 unit cells
- Disorder threshold:   _crit   > 15% (strong topological protection)

Use symbolic math for   (r) Fourier expansions. Export all results as
    JSON + STL.
Pure Maxwell theory  o n l y no  quantum mechanics, no experimental
    fitting.
```

---

## 0.5 5. Success Criteria

### 0.5.1 Minimum Viable Result (MVR)

**Within 1-2 months**:

- **Band Structure Solver**: Plane wave expansion method working for square lattice
- **Band Gap Identified**: / > 5
- **Validation**: Reproduce literature results for test structures

**Deliverable**: Photonic band structure code + plots

### 0.5.2 Strong Result

**Within 4-5 months**:

- **Topological Design**: Photonic Chern insulator with C = 1 constructed
- **Edge States**: Computed and visualized, unidirectional propagation verified
- **Optimization**: Band gap / > 10
- **Robustness**: Edge states survive / = 15

**Metrics**: Certificate exported with exact C = 1, edge mode dispersion, field patterns

### 0.5.3 Publication-Quality Result

**Within 6 months**:

- **Complete Design**: Fabrication-ready STL file for 3D printing
- **Materials Specification**: TiO or Si rods in polymer matrix
- **Operating Frequency**: Optimized for telecom (1.5 m) or mid-IR (10 m)
- **Experimental Predictions**: FDTD simulations confirm topological protection
- **Multiple Designs**: C = 1, 2 Chern insulators + topological photonics

**Publications**: "Pure-Thought Design of Topological Photonic Crystals"

---

## 0.6 6. Verification Protocol

Standard checks: Chern number re-computation, edge state counting, disorder simulations, FDTD cross-validation.

---

## 0.7 7. Resources Milestones

**Key References**:

- Haldane Raghu (2008): "Possible Realization of Directional Optical Waveguides"
- Wang et al. (2009): "Observation of Unidirectional Backscattering-Immune Topological States"
- Lu et al. (2014): "Topological Photonics" (Nature Photonics review)

**Milestones**:

- Month 1: Band solver validated
- Month 3: C=1 design complete
- Month 5: Edge states + robustness certified
- Month 6: STL exported, fabrication protocol written

---

### 0.8 8. Extensions

- **3D Topological Photonics**: Weyl points in 3D crystals
- **Higher-Order Topology**: Corner states in 2D photonics
- **Nonlinear Photonics**: Topology + $^{2}/^{3}$ interactions

---

**End of PRD 11**