Challenge 07: Extremal Higher-Dimensional CFTs with Stress Tensor

Pure Thought AI Challenge 07

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: Quantum Gravity Particle Physics
Difficulty: Medium-High
Timeline: 6-12 months
Prerequisites: Conformal field theory, conformal bootstrap, AdS/CFT

0.1 Problem Statement

0.1.1 Scientific Context

The conformal bootstrap has produced rigorous universal bounds on conformal field theory data using only crossing symmetry and unitarity. For large central charge CFTs with sparse spectra—theories dual to Einstein gravity in Anti-de Sitter space—we can ask foundational questions about the maximum possible spectral gaps.

These questions directly constrain pure quantum gravity. If a CFT has only the stress tensor and a large gap to the next operator, its holographic dual is pure Einstein gravity without additional light fields.

0.1.2 The Core Question

What is the maximum possible gap to the first non-conserved operator in a unitary CFT$_d withonlythestresstensor(anditsdescendants)$?

More precisely:

- **Given dimension d and central charge c**$_T$
- **Assuming the operator spectrum contains: identity, stress tensor T**$_{,andpossiblyhigher-spinconservedcurrents}$
- **What is the maximum** $_gaptothefirstnon-conservedprimaryoperator$?

Related questions:

- **Can higher-spin conserved currents (J = 4, 6, ...) exist without non-conserved operators below some gap?**
- **What are universal lower bounds on operator dimensions?**

0.1.3 Why This Matters

- **AdS/CFT: Bounds on** $_gapconstrainpureEinsteingravitytheories$
- **Universal results: Apply to all holographic CFTs regardless of details**
- **Rigorous: Derived via extremal functional method with certificates**
- **Testable: Compare to known CFTs (free theories, holographic examples)**
- **Gravitational constraints: Proves Einstein gravity is "special"**

0.2 Mathematical Formulation

0.2.1 Problem Definition

Consider a unitary CFT in d spacetime dimensions with stress tensor $T_{.}$
Stress tensor 4-point function:

```
G( x  , x  , x  , x  ) =  T ( x  ) T( x  ) T( x  ) T( x  )
```

Conformal block decomposition:

```
G =  _ {  ,J} C  _{T,T,O_{  ,J}} G_{  ,J}(u,v)
```

where:

- (, J) = conformal dimension and spin of exchanged operator
- $C_{T,T,O} = OPEcoefficient$
- $G_{,J}(u,v) = conformalblock$
- u, v = conformal cross-ratios

Operator content assumptions:

- Identity: = 0
- Stress tensor: = d, J = 2 (conserved)
- Possibly higher-spin currents: = d, J = 4, 6, 8, ...
- Gap assumption: No primary operators with d < $<_{g}ap$

Constraints:

- Crossing symmetry:

```
G(u,v) = G(v,u)
```

- Unitarity: All OPE coefficients squared C² 0
- Ward identities: Conservation of $T_{imposesrelations}$
- Gap assumption: Spectrum restricted as above

0.2.2 Optimization Formulation

Upper bound on gap (impossibility):
Find an extremal functional (, J) such that:

```
 _ {  ,J}   (  , J) [crossing equation for G_{  ,J}] < 0
```

with:

- (, J) 0 for all (identity, T, higher-spin currents)
- (, J) 0 for $<_{g}ap(excludedregion)$

This proves no CFT with gap $_{g}apexists.$
Lower bound on gap (construction):
Exhibit an explicit CFT (or holographic model) with gap $=_{g}apachievingthebound.$

0.2.3 Certificate of Correctness

If claiming $_{g}apismaximum$:

- **Extremal functional: (, J) proving** $> {}_{g}apisimpossible$
- **Verification: Check 0 on allowed region**
- **Verification: Compute applied to crossing equation, verify** < 0
- **Verification: Check all derivatives/positivity conditions**

If claiming $_{g}apisachievable$:

- **Explicit CFT: Construct theory with this gap**
- **Or: Holographic model (bulk theory) predicting this gap**
- **Verification: Compute spectrum, verify gap**

0.3 Implementation Approach

0.3.1 Phase 1: Conformal Blocks for Stress Tensor (Months 1-2)

Implement $\langle$**TTTT**$\rangle$ **conformal blocks:**

```
import numpy as np
from mpmath import mp, hyp2f1
mp.dps = 100

def stress_tensor_conformal_block(Delta, J, u, v, d=3):
    """
    Compute conformal block for stress tensor 4-point function

      T   T T   T    conformal block for exchange of operator (   , J)

    Uses Casimir differential equation or recursion relations
    """
    # For external operators with  _ext  = d (stress tensor)
    # Internal operator: (  , J)

    # Simplified: use hypergeometric functions
    # Full implementation requires solving Casimir equation

    z, zbar = conformal_cross_ratios_to_z(u, v)

    # Prefactor from  three-point function kinematics
    prefactor = compute_three_point_prefactor(d, d, Delta, J)

    # Hypergeometric piece (schematic)
    block = prefactor * z**(Delta/2) * zbar**(Delta/2) * \
            hyp2f1(Delta/2, Delta/2, Delta, z) * \
            hyp2f1(Delta/2, Delta/2, Delta, zbar)
```

```python
    return block

def conformal_cross_ratios_to_z(u, v):
    """
    Convert (u,v) cross-ratios to (z, z )

    u = z z , v = (1-z)(1-z )
    """
    # Solve for z, z
    discriminant = np.sqrt(v**2 - 2*v*(u+1) + (u-1)**2)
    z = (v - discriminant) / (2*v - 2)
    zbar = (v + discriminant) / (2*v - 2)

    return z, zbar

def compute_three_point_prefactor(Delta1, Delta2, Delta3, J):
    """
    Compute kinematic prefactor from  T  T  O  three-point function

    Depends on spacetime dimension d and operator quantum numbers
    """
    # Implement using conformal representation theory
    # Depends on structure constants
    pass
```

Implement recursion relations:

```python
def casimir_differential_equation(g, Delta, J, d):
    """
    Conformal blocks satisfy Casimir differential equation

    D g(z, z ) = C_2( , J) g(z, z )

    where D is Casimir differential operator
    """
    # Second-order PDE in z, z
    # Can be solved via series expansion or numerically
    pass

def recursion_relation_conformal_block(Delta, J, n_max):
    """
    Use Zamolodchikov-like recursion to build conformal blocks
    """
    coefficients = np.zeros(n_max)
    coefficients[0] = 1  # Normalization

    for n in range(1, n_max):
        # Recursion: a_n = f(a_{n-1}, a_{n-2}, , J)
        coefficients[n] = compute_recursion_coefficient(
            coefficients[n-1], coefficients[n-2], Delta, J
        )

    return coefficients
```

0.3.2 Phase 2: Crossing Equation (Months 2-3)

Set up crossing symmetry:

```
def crossing_equation_TTTT(spectrum, u_point, v_point, d=3):
    """
    Crossing:    T    T    T    T    =    T    T    T    T

       C  (  ,J) G_{  ,J}(u,v) =    C  (  ,J) G_{  ,J}(v,u)
    """
    # s-channel sum
    s_channel = 0
    for Delta, J, C_squared in spectrum:
        block_s = stress_tensor_conformal_block(Delta, J, u_point,
            v_point, d)
        s_channel += C_squared * block_s

    # t-channel sum (swap u     v)
    t_channel = 0
    for Delta, J, C_squared in spectrum:
        block_t = stress_tensor_conformal_block(Delta, J, v_point,
            u_point, d)
        t_channel += C_squared * block_t

    # Crossing equation residual
    residual = s_channel - t_channel

    return residual

def setup_crossing_matrix(Delta_grid, J_values, test_points, d=3):
    """
    Discretize crossing equation on grid

    Returns matrix M such that M    C   = 0 enforces crossing
    """
    n_operators = len(Delta_grid) * len(J_values)
    n_test_points = len(test_points)

    M = np.zeros((n_test_points, n_operators))

    for i, (u, v) in enumerate(test_points):
        for j, (Delta, J) in enumerate(product(Delta_grid, J_values)):
            # s-channel block
            block_s = stress_tensor_conformal_block(Delta, J, u, v, d)
            # t-channel block
            block_t = stress_tensor_conformal_block(Delta, J, v, u, d)

            M[i, j] = block_s - block_t

    return M
```

0.3.3 Phase 3: Ward Identities Conservation (Months 3-4)

Stress tensor conservation constraints:

```
def stress_tensor_ward_identity(correlator, d):
```

```
    """
    Conservation:    ^    T_     = 0

    Imposes constraints on OPE coefficients and form of correlator
    """
    # For   T    T   : fixed by conformal symmetry
    # For   T   T  T   : constrains form
    # For   T   T T  T   : additional sum rules

    # Ward identity: certain derivative of correlator vanishes
    #     _       T ^    ( x   ) T( x   ) T( x   ) T( x   )    = contact
       terms

    pass

def implement_conservation_constraints(OPE_data):
    """
    Conservation of T implies:
    - OPE T    T contains only operators with specific properties
    - Recursion relations among OPE coefficients
    """
    # Conserved spin-J current:    = d + J - 2
    # For stress tensor (J=2):    = d

    constraints = []

    # T    T OPE must contain identity + T + higher-spins or gaps
    for Delta, J in OPE_data:
        if J % 2 == 1:  # Odd spin forbidden by symmetry
            constraints.append((Delta, J, 'forbidden'))
        elif J > 0 and Delta < d:  # Sub-leading twist
            constraints.append((Delta, J, 'forbidden'))

    return constraints
```

0.3.4 Phase 4: Extremal Functional Method (Months 4-8)

Linear functional approach:

```
def extremal_functional_method(Delta_gap, J_max, d=3, n_derivatives=4):
    """
    Find extremal functional   (  , J) proving  _gap  is impossible

    The functional must:
    1. Be positive on allowed operators (ID, T, higher-spins)
    2. Be positive for    <  _gap  (excluded region)
    3. Make crossing equation negative (proving inconsistency)
    """
    # Functional is determined by its action on conformal blocks
    # Parametrize by derivatives at crossing-symmetric point

    #    is linear functional:   [G] =  _n   _n    _z ^n G|_{z=z  =1/2}

    alpha_coeffs = cp.Variable(n_derivatives)

    constraints = []
```

```

    # 1. POSITIVITY on identity
    alpha_identity = evaluate_functional_on_identity(alpha_coeffs)
    constraints.append(alpha_identity >= 0)

    # 2. POSITIVITY on stress tensor
    alpha_T = evaluate_functional_on_stress_tensor(alpha_coeffs, d)
    constraints.append(alpha_T >= 0)

    # 3. POSITIVITY on allowed higher-spin currents (if any)
    for J in [4, 6, 8]:  # Spin-4, 6, 8 currents
        if allow_higher_spins:
            alpha_J = evaluate_functional_on_current(alpha_coeffs, d, J)
            constraints.append(alpha_J >= 0)

    # 4. POSITIVITY for   <  _gap  (excluded region)
    Delta_test_points = np.linspace(d+0.01, Delta_gap-0.01, 50)
    for Delta_test in Delta_test_points:
        for J_test in [0, 2, 4]:
            alpha_test = evaluate_functional_on_block(
                alpha_coeffs, Delta_test, J_test, d
            )
            constraints.append(alpha_test >= 0)

    # 5. NORMALIZATION: make crossing equation negative
    #   [crossing equation] < 0
    alpha_crossing = evaluate_functional_on_crossing(alpha_coeffs, d)
    constraints.append(alpha_crossing == -1)  # Normalize to -1

    # Solve feasibility problem
    problem = cp.Problem(cp.Minimize(0), constraints)
    problem.solve(solver=cp.MOSEK)

    if problem.status == 'optimal':
        return {
            'status': 'bound_proven',
            'Delta_gap_max': Delta_gap,
            'functional': alpha_coeffs.value
        }
    else:
        return {
            'status': 'gap_allowed',
            'Delta_gap': Delta_gap
        }

def evaluate_functional_on_block(alpha_coeffs, Delta, J, d):
    """
    Evaluate functional   on conformal block G_{  ,J}

      [G] =  _n   _n  (  _z ^n G)|_{z=z  =1/2}
    """
    # Compute derivatives of conformal block
    derivatives = []
    z_sym = 0.5

    for n in range(len(alpha_coeffs)):
```

```
        deriv_n = compute_nth_derivative_block(Delta, J, z_sym, n, d)
        derivatives.append(deriv_n)

    #   [G] = dot product
    alpha_value = np.dot(alpha_coeffs, derivatives)

    return alpha_value
```

0.3.5 Phase 5: Binary Search for Maximum Gap (Months 8-10)

```
def binary_search_maximum_gap(d=3, J_max=8):
    """
    Find maximum  _gap  via binary search

    For each candidate gap, check if extremal functional exists
    """
    Delta_min = d + 0.01  # Just above stress tensor
    Delta_max = 3*d  # Conservative upper bound

    tolerance = 0.01

    while Delta_max - Delta_min > tolerance:
        Delta_mid = (Delta_min + Delta_max) / 2

        print(f"Testing gap = {Delta_mid:.3f}")

        result = extremal_functional_method(Delta_mid, J_max, d)

        if result['status'] == 'bound_proven':
            # Gap this large is impossible
            Delta_max = Delta_mid
            print(f"      Gap {Delta_mid:.3f} ruled out")
        else:
            # Gap this large might be allowed
            Delta_min = Delta_mid
            print(f"      Gap {Delta_mid:.3f} allowed")

    return {
        'max_gap': Delta_min,
        'dimension': d,
        'status': 'converged'
    }
```

0.3.6 Phase 6: Verification Comparison (Months 10-12)

```
def verify_gap_bound(Delta_gap_claimed, extremal_functional, d):
    """
    Verify claimed maximum gap
    """
    print(f"Verifying maximum gap  _gap  = {Delta_gap_claimed} in
       d={d}")

    # 1. Check functional is positive on identity
```

```
    alpha_ID = evaluate_functional_on_identity(extremal_functional)
    assert alpha_ID >= -1e-10, f"Functional negative on identity: {alpha_ID}"

    # 2. Check functional is positive on stress tensor
    alpha_T = evaluate_functional_on_stress_tensor(extremal_functional, d)
    assert alpha_T >= -1e-10, f"Functional negative on T: {alpha_T}"

    # 3. Check functional is positive for   <  _gap
    for Delta_test in np.linspace(d+0.1, Delta_gap_claimed-0.1, 100):
        alpha_test = evaluate_functional_on_block(extremal_functional, Delta_test, 0, d)
        assert alpha_test >= -1e-8, f"Functional negative at ={Delta_test}: {alpha_test}"

    # 4. Check functional makes crossing negative
    alpha_crossing = evaluate_functional_on_crossing(extremal_functional, d)
    assert alpha_crossing < -1e-6, f"Crossing not negative: {alpha_crossing}"

    print("All checks passed! Bound verified.")

    # 5. Compare to known CFTs
    compare_to_known_cfts(Delta_gap_claimed, d)

    return True

def compare_to_known_cfts(Delta_gap_bound, d):
    """
    Compare bound to known CFT examples
    """
    known_cfts = {
        'd=3': {
            'Free scalar': {'gap': 2.0, 'description': '   =0'},
            'Ising CFT': {'gap': 0.5181489, 'description': ' primary'},
            'O(N) model': {'gap': 'varies', 'description': ' ^i primary'},
        }
    }

    if f'd={d}' in known_cfts:
        print(f"\nComparison with known d={d} CFTs:")
        for name, data in known_cfts[f'd={d}'].items():
            if isinstance(data['gap'], (int, float)):
                if data['gap'] < Delta_gap_bound:
                    print(f"    {name}: gap={data['gap']:.4f} < {Delta_gap_bound:.4f} (consistent)")
                else:
                    print(f"    {name}: gap={data['gap']:.4f} > {Delta_gap_bound:.4f} (VIOLATION!)")
```

0.4 Example Starting Prompt

```
I need you to implement the conformal bootstrap for stress tensor 4-point functions
to derive universal bounds on operator gaps in CFTs.

GOAL: Find the maximum gap  _gap  to the first non-conserved operator in a d=3 CFT
with only the identity and stress tensor.

PHASE 1 - Build conformal blocks:
1. Implement the conformal block G_{ ,J}(z, z ) for   TTTT correlator

2. For d=3, implement blocks for:
   - Identity exchange
   - Stress tensor exchange
   - Generic scalar exchange ( , J=0)
   - Generic spin-2 exchange ( , J=2)

3. Verify blocks satisfy Casimir differential equation

PHASE 2 - Crossing symmetry:
4. Write down the crossing equation: G(u,v) = G(v,u)

5. Discretize at test points and set up matrix equation

PHASE 3 - Linear functional:
6. Parametrize functional   by derivatives at crossing-symmetric point

7. Impose positivity:
   -  [ID]   0
   -  [T]   0
   -  [G_{ ,J}]   0 for all   <  _gap

8. Impose  [crossing equation] < 0 (proves inconsistency)

PHASE 4 - Optimization:
9. Formulate as linear program and solve using cvxpy + MOSEK

10. Binary search to find maximum  _gap  where functional exists

PHASE 5 - Verification:
11. Extract extremal functional and verify all positivity conditions

12. Compare bound to known CFTs (Ising, free theories)

13. Plot functional action on conformal block spectrum

Please implement with high precision and cross-check against bootstrap literature.
```

0.5 Success Criteria

0.5.1 Minimum Viable Result (6 months)

Bootstrap infrastructure:

- **Stress tensor conformal blocks implemented for d=3**
- **Crossing equation verified numerically**
- **Linear functional method working**

First bound:

- **Maximum gap bound in d=3 obtained**
- **Extremal functional extracted and verified**
- **Comparison with Ising CFT confirms consistency**

0.5.2 Strong Result (9 months)

Multiple dimensions:

- **Bounds obtained for d=3, 4, 5**
- **Universal patterns identified**
- **With and without higher-spin currents**

Rigorous certificates:

- **All extremal functionals verified**
- **Positivity checked to high precision**
- **Comparison with holographic predictions**

0.5.3 Publication-Quality Result (12 months)

Comprehensive classification:

- **Complete gap bounds for d=2-6**
- **Phase diagram: (d, c**$_T$)ß$maximumgap$
- **Identification of "allowed" vs "forbidden" regions**

Formal verification:

- **Functional positivity certified**
- **Lean formalization of crossing symmetry**
- **Publication with numerical data repository**

0.6 Verification Protocol

```
def verify_extremal_functional(alpha, Delta_gap, d):
    """
    Comprehensive verification of extremal functional
    """
    # 1. Positivity on identity
    assert alpha_on_identity(alpha) >= 0

    # 2. Positivity on stress tensor
    assert alpha_on_stress_tensor(alpha, d) >= 0

    # 3. Positivity for    <   _gap
    for Delta in np.linspace(d, Delta_gap, 200):
        for J in [0, 2, 4]:
            assert alpha_on_block(alpha, Delta, J, d) >= -1e-10

    # 4. Negativity on crossing
    assert alpha_on_crossing(alpha, d) < -1e-6

    # 5. Consistency with known CFTs
    for cft_name, cft_gap in known_gaps(d).items():
        assert cft_gap <= Delta_gap, f"{cft_name} violates bound!"

    return "BOUND VERIFIED"
```

0.7 Milestone Checklist

Conformal blocks for ⟨TTTT⟩ implemented (d=3)

Casimir equation verified

Crossing symmetry checked numerically

Ward identities for T conservation implemented

Linear functional method coded

First gap bound obtained (d=3)

Extremal functional positivity verified

Binary search for maximum gap working

Comparison with Ising CFT done

Results for d=4,5 obtained

Holographic comparison completed

Publication draft with data

Next Steps: Start by implementing conformal blocks for identity and stress tensor exchange in d=3. Verify crossing symmetry numerically before attempting the linear functional optimization.