Challenge 07: Extremal Higher-Dimensional CFTs

Conformal Bootstrap Bounds on OPE Data in $d > 2$ Dimensions

Domain:	Quantum Gravity & CFT
Difficulty:	High
Timeline:	8–12 months
Prerequisites:	Conformal field theory, bootstrap methods, semidefinite programming (SDP) optimization

Pure Thought AI Challenges
January 19, 2026

Contents

1 Executive Summary **3**

2 Scientific Context and Motivation **3**
2.1 Conformal Bootstrap in $d > 2$ Dimensions . . . 3
2.2 The Operator Product Expansion . . . 4
2.3 The TTTT Correlator Bootstrap . . . 4
2.4 Stress Tensor OPE Structure . . . 5
2.5 Central Charges a and c in 4d . . . 5
2.6 Unitarity Bounds on Dimensions . . . 5
2.7 The Core Questions . . . 6

3 Mathematical Formulation **6**
3.1 Conformal Blocks in d Dimensions . . . 6
3.1.1 Closed-Form Expressions in Even Dimensions . . . 6
3.1.2 Casimir Differential Equation . . . 7
3.2 Crossing Equations for Stress Tensor Four-Point Function . . . 7
3.3 OPE Decomposition and Selection Rules . . . 8
3.4 The Extremal Functional Method . . . 8
3.5 Central Charge Bounds: a/c from TTTT Bootstrap . . . 9
3.6 SDP Formulation for Bootstrap Bounds . . . 9

4 Implementation Approach **9**
4.1 Phase 1: Conformal Block Computation (Months 1–2) . . . 9
4.2 Phase 2: Crossing Matrix Setup (Months 2–3) . . . 12
4.3 Phase 3: SDP Formulation for Bounds (Months 3–5) . . . 14
4.4 Phase 4: Extremal Functional Method (Months 5–7) . . . 16
4.5 Phase 5: TTTT-Specific Implementation (Months 6–8) . . . 18
4.6 Phase 6: Certificate Generation (Months 8–10) . . . 19

5 Research Directions **22**
5.1 Direction 1: Gap Bounds in $d = 3, 4, 6$ . . . 22
5.2 Direction 2: OPE Coefficient Bounds . . . 22
5.3 Direction 3: Comparison with Known CFTs . . . 23
5.4 Direction 4: Holographic Interpretation . . . 23
5.5 Direction 5: Improved a/c Bounds . . . 24
5.6 Direction 6: Mixed Correlator Bootstrap . . . 24

6 Success Criteria **24**
6.1 Minimum Viable Result (4–5 months) . . . 24
6.2 Strong Result (6–8 months) . . . 25
6.3 Publication-Quality Result (10–12 months) . . . 25

7 Verification Protocol **25**
7.1 Conformal Block Verification . . . 25
7.2 SDP Solution Verification . . . 26
7.3 Literature Comparison . . . 27

8 Common Pitfalls and Mitigations **28**
8.1 Numerical Precision in Conformal Blocks . . . 28
8.2 SDP Solver Issues . . . 28
8.3 Discretization Artifacts . . . 29
8.4 Tensor Structure Errors . . . 29
8.5 Convergence of Derivative Expansion . . . 30

9 Resources and References **30**
9.1 Essential Papers . . . 30
9.2 Code Libraries . . . 30
9.3 Mathematical Background . . . 31

10 Milestone Checklist **31**
10.1 Infrastructure (Months 1–2) . . . 31
10.2 Scalar Bootstrap (Months 2–3) . . . 31
10.3 OPE Bounds (Months 3–4) . . . 31
10.4 TTTT Bootstrap (Months 4–6) . . . 32
10.5 a/c Bounds (Months 6–8) . . . 32
10.6 Extremal Analysis (Months 8–10) . . . 32
10.7 Publication (Months 10–12) . . . 32

11 Conclusion **32**

A Conformal Block Technical Details **33**
A.1 Zamolodchikov Recursion . . . 33
A.2 Casimir Operator . . . 33

B Tensor Structures for TTTT **33**
B.1 Embedding Space . . . 33
B.2 Building Blocks . . . 34
B.3 TTTT Structures in 4d . . . 34

C SDP Implementation Details **34**
C.1 Polynomial Positivity via SOS . . . 34

1 Executive Summary

This challenge addresses the systematic derivation of **rigorous bounds on conformal field theory data** in spacetime dimensions $d > 2$ using the **conformal bootstrap**. Unlike two-dimensional CFTs where infinite-dimensional Virasoro symmetry provides extraordinary control, higher-dimensional CFTs have finite-dimensional conformal groups, making the bootstrap both more challenging and more universal in its predictions.

The central objects of study are the **OPE coefficients** $\lambda_{\mathcal{O}_1\mathcal{O}_2\mathcal{O}_3}$ and **operator dimensions** $\Delta_{\mathcal{O}}$ that characterize any CFT. By exploiting **crossing symmetry** of four-point correlation functions and **unitarity**, the bootstrap produces non-perturbative constraints that apply to *all* CFTs in a given dimension—including strongly coupled theories inaccessible to perturbation theory.

Analysis Note

The conformal bootstrap has achieved remarkable success in recent years, most notably providing the world's most precise determination of critical exponents in the 3D Ising model—more accurate than any Monte Carlo simulation or experiment. This challenge extends these methods to **stress tensor four-point functions** ($\langle TTTT \rangle$), which encode central charges and gravitational couplings in holographic theories.

Physical Insight

Why Higher Dimensions Matter:

- $d = 3$: Describes critical phenomena in condensed matter (Ising, $O(N)$ models)
- $d = 4$: Directly relevant for particle physics and gauge theories
- $d = 6$: Contains the mysterious $(2, 0)$ superconformal theory
- General d: Holographic CFTs dual to quantum gravity in AdS_{d+1}

The specific focus of this challenge is the **TTTT bootstrap**—the crossing equations derived from the four-point function of stress tensors. This provides bounds on the central charges a and c in 4d CFTs, constraints on the gap to the first non-identity operator in the $T \times T$ OPE, and universal features of holographic theories.

2 Scientific Context and Motivation

2.1 Conformal Bootstrap in $d > 2$ Dimensions

The **conformal bootstrap** is a program to classify and constrain CFTs using only consistency conditions:

1. **Conformal symmetry:** The theory is invariant under the conformal group $SO(d + 1, 1)$
2. **Operator product expansion (OPE):** Products of local operators can be expanded in a complete basis
3. **Crossing symmetry:** OPE in different channels must give identical four-point functions
4. **Unitarity:** Physical states have positive norm (equivalent to reflection positivity)

Definition 2.1 (Conformal Primary Operator)**.** A local operator $\mathcal{O}(x)$ is a **conformal primary** with scaling dimension Δ and spin ℓ if:

$$[D, \mathcal{O}(0)] = \Delta\, \mathcal{O}(0) \tag{1}$$

$$[M_{\mu\nu}, \mathcal{O}(0)] = S_{\mu\nu}\, \mathcal{O}(0) \quad \text{(spin-}\ell\text{ representation)} \tag{2}$$

$$[K_\mu, \mathcal{O}(0)] = 0 \tag{3}$$

where D is the dilatation generator, $M_{\mu\nu}$ are Lorentz rotations, and K_μ are special conformal generators.

2.2 The Operator Product Expansion

The OPE is the fundamental structure underlying the bootstrap:

$$\boxed{\mathcal{O}_1(x)\mathcal{O}_2(0) = \sum_{\mathcal{O}} \lambda_{12\mathcal{O}}\, C(x, \partial)\, \mathcal{O}(0)} \tag{4}$$

where:

- $\lambda_{12\mathcal{O}}$ is the **OPE coefficient** (structure constant)
- $C(x, \partial)$ is a differential operator fixed entirely by conformal symmetry
- The sum runs over all conformal primaries $\mathcal{O}$ in the theory

Physical Insight

CFT Data: A CFT in d dimensions is completely specified by:

1. The spectrum of primary operators $\{\mathcal{O}_i\}$ with their dimensions Δ_i and spins ℓ_i
2. All OPE coefficients λ_{ijk}

The bootstrap constrains this "CFT data" without ever computing a Feynman diagram or path integral.

2.3 The TTTT Correlator Bootstrap

The four-point function of stress tensors is the central object for this challenge:

$$\langle T_{\mu_1\nu_1}(x_1) T_{\mu_2\nu_2}(x_2) T_{\mu_3\nu_3}(x_3) T_{\mu_4\nu_4}(x_4) \rangle \tag{5}$$

Physical Insight

Why TTTT?

- The stress tensor exists in *every* CFT—results are universal
- OPE coefficients in $T \times T$ encode central charges and coupling to gravity
- Holographic interpretation: TTTT correlator $\leftrightarrow$ graviton scattering in AdS
- The a/c ratio distinguishes theories and has Swampland implications

The correlator decomposes into tensor structures and conformal blocks:

$$\langle TTTT \rangle = \sum_I \mathcal{T}_I(x_i, \partial) \sum_{\mathcal{O}} \lambda^{(I)}_{TT\mathcal{O}} \lambda^{(I)}_{TT\mathcal{O}} G^{(I)}_{\Delta_{\mathcal{O}}, \ell_{\mathcal{O}}}(u, v) \tag{6}$$

where $\mathcal{T}_I$ are kinematic tensor structures and $G^{(I)}$ are conformal blocks.

2.4 Stress Tensor OPE Structure

The $T \times T$ OPE takes the schematic form:

$$T_{\mu\nu} \times T_{\rho\sigma} \sim \mathbf{1} + T_{\alpha\beta} + \mathcal{O}^{(\text{scalar})} + \mathcal{O}^{(\text{spin-4})} + \dots \tag{7}$$

Definition 2.2 (Operators in $T \times T$ OPE)**.** The operators appearing in the stress tensor OPE include:

- **Identity 1**: Always present, coefficient fixed by Ward identity
- **Stress tensor** T: Present with coefficient related to central charge
- **Scalars**: ϕ^2-type operators with $\Delta \geq d - 2$
- **Spin-4**: Double-trace $[TT]_0$ with $\Delta \approx 2d$
- **Higher spins**: Only even spins by Bose symmetry

2.5 Central Charges a and c in 4d

In four dimensions, there are two independent central charges appearing in the trace anomaly:

Theorem 2.1 (Weyl Anomaly in 4d)**.** On a curved background, the stress tensor trace satisfies:

$$\langle T^\mu_\mu \rangle = \frac{c}{16\pi^2} W_{\mu\nu\rho\sigma} W^{\mu\nu\rho\sigma} - \frac{a}{16\pi^2} E_4 \tag{8}$$

where $W_{\mu\nu\rho\sigma}$ is the Weyl tensor and $E_4 = R^2_{\mu\nu\rho\sigma} - 4R^2_{\mu\nu} + R^2$ is the Euler density.

Analysis Note

The central charges have important physical interpretations:

- c: Appears in $\langle TT \rangle$ two-point function normalization
- a: Counts degrees of freedom; decreases under RG flow (a-theorem)
- a/c: A universal ratio constrained by causality and unitarity

For reference, central charges of known theories:

Theory	a	c
Free real scalar	$\frac{1}{360}$	$\frac{1}{120}$
Free Dirac fermion	$\frac{11}{360}$	$\frac{1}{20}$
Free vector (Maxwell)	$\frac{31}{180}$	$\frac{1}{10}$
$\mathcal{N} = 4$ SYM ($SU(N)$)	$\frac{N^2-1}{4}$	$\frac{N^2-1}{4}$

2.6 Unitarity Bounds on Dimensions

Unitarity (positive norm of states) imposes lower bounds on operator dimensions:

Theorem 2.2 (Unitarity Bounds)**.** In a unitary CFT in d dimensions, conformal primary operators satisfy:

$$\text{Scalar } (\ell = 0): \quad \Delta \geq \frac{d-2}{2} \quad \text{(free scalar at saturation)} \tag{9}$$

$$\text{Spin-}\ell \ (\ell \geq 1): \quad \Delta \geq d + \ell - 2 \quad \text{(conserved current at saturation)} \tag{10}$$

Critical Consideration

Subtlety for Conserved Currents: When $\Delta = d + \ell - 2$ is saturated, the operator is a conserved current satisfying $\partial \cdot J = 0$. These have null descendants requiring special treatment in the bootstrap. The stress tensor with $\Delta = d$, $\ell = 2$ is the prime example.

2.7 The Core Questions

Central Research Questions

Q1: What are the allowed ranges for a/c in 4d CFTs?

- Known lower bound: $a/c \geq 1/3$ (Hofman-Maldacena, 2008)
- Known upper bound: $a/c \leq 31/18$
- Can the bootstrap sharpen these bounds?

Q2: What is the maximum gap Δ_{gap} to the first scalar in $T \times T$?

- In free theory: $\Delta_{\text{gap}} = d - 2$ (operator ϕ^2)
- In holographic theories: what is the bound?

Q3: Can TTTT bootstrap isolate specific CFTs?

- Does $\mathcal{N} = 4$ SYM sit at a kink?
- Can we identify extremal theories saturating bounds?

3 Mathematical Formulation

3.1 Conformal Blocks in d Dimensions

Conformal symmetry fixes the form of four-point functions up to functions of cross-ratios:

$$\langle \mathcal{O}_1(x_1)\mathcal{O}_2(x_2)\mathcal{O}_3(x_3)\mathcal{O}_4(x_4)\rangle = \frac{g(u,v)}{x_{12}^{\Delta_1+\Delta_2} x_{34}^{\Delta_3+\Delta_4}} \cdot (\text{tensor structure}) \tag{11}$$

where the **conformal cross-ratios** are:

$$u = \frac{x_{12}^2 x_{34}^2}{x_{13}^2 x_{24}^2}, \quad v = \frac{x_{14}^2 x_{23}^2}{x_{13}^2 x_{24}^2} \tag{12}$$

Definition 3.1 (Conformal Block)**.** The **conformal block** $G^{(d)}_{\Delta,\ell}(u,v)$ is the contribution to the four-point function from a single conformal primary $\mathcal{O}$ with dimension Δ and spin ℓ, including all descendants:

$$g(u,v) = \sum_{\mathcal{O}} \lambda_{12\mathcal{O}}\lambda_{34\mathcal{O}}\, G^{(d)}_{\Delta_{\mathcal{O}},\ell_{\mathcal{O}}}(u,v) \tag{13}$$

3.1.1 Closed-Form Expressions in Even Dimensions

In even dimensions, conformal blocks have closed-form expressions. For $d = 4$:

$$G^{(4)}_{\Delta,\ell}(u,v) = \frac{z\bar{z}}{z-\bar{z}}\left[k_{\Delta+\ell}(z)k_{\Delta-\ell-2}(\bar{z}) - (z \leftrightarrow \bar{z})\right] \tag{14}$$

where $u = z\bar{z}$, $v = (1-z)(1-\bar{z})$, and the building blocks are:

$$k_\beta(z) = z^{\beta/2}\, {}_2F_1\left(\frac{\beta}{2}, \frac{\beta}{2}, \beta; z\right) \tag{15}$$

For $d = 6$:

$$G^{(6)}_{\Delta,\ell}(u,v) = \frac{(z\bar{z})^2}{(z-\bar{z})^3} \sum_{\text{perms}} (\pm) k_{\alpha_1}(z) k_{\alpha_2}(\bar{z}) \tag{16}$$

with appropriate index combinations.

Analysis Note

In odd dimensions (e.g., $d = 3$), there is no known closed form. Conformal blocks must be computed via:

1. Recursion relations (Zamolodchikov-type)
2. Series expansion in radial coordinates
3. Casimir differential equation methods

3.1.2 Casimir Differential Equation

Conformal blocks are eigenfunctions of the quadratic Casimir operator:

$$\mathcal{D}^{(d)} G^{(d)}_{\Delta,\ell}(u,v) = c_{\Delta,\ell}\, G^{(d)}_{\Delta,\ell}(u,v) \tag{17}$$

where the eigenvalue is:

$$c_{\Delta,\ell} = \Delta(\Delta - d) + \ell(\ell + d - 2) \tag{18}$$

The Casimir operator in terms of $z, \bar{z}$ coordinates:

$$\mathcal{D}^{(d)} = D_z + D_{\bar{z}} + \frac{d-2}{z-\bar{z}}\left[(1-z)z\partial_z - (1-\bar{z})\bar{z}\partial_{\bar{z}}\right] \tag{19}$$

where $D_z = (1-z)z^2\partial_z^2 - z^2\partial_z$.

3.2 Crossing Equations for Stress Tensor Four-Point Function

The OPE can be performed in different channels. For $\langle T_1 T_2 T_3 T_4 \rangle$:
***s*-channel:** OPE of $(T_1 T_2)$ and $(T_3 T_4)$

$$\langle TTTT \rangle^{(s)} = \sum_{\mathcal{O}} |\lambda_{TT\mathcal{O}}|^2 G^{(s)}_{\Delta,\ell}(u,v) \tag{20}$$

***t*-channel:** OPE of $(T_1 T_4)$ and $(T_2 T_3)$

$$\langle TTTT \rangle^{(t)} = \sum_{\mathcal{O}'} |\lambda_{TT\mathcal{O}'}|^2 G^{(t)}_{\Delta',\ell'}(v,u) \tag{21}$$

Crossing symmetry requires:

$$\boxed{\sum_{\mathcal{O}} |\lambda_{TT\mathcal{O}}|^2 \left[G^{(s)}_{\Delta,\ell}(u,v) - G^{(t)}_{\Delta,\ell}(v,u)\right] = 0} \tag{22}$$

3.3 OPE Decomposition and Selection Rules

Theorem 3.1 (Selection Rules for $T \times T$ OPE)**.** In the stress tensor OPE:

1. Only **even spins** $\ell = 0, 2, 4, \ldots$ appear (Bose symmetry)
2. The **identity** appears with coefficient related to c
3. The **stress tensor** T appears with coefficient encoding both a and c
4. **Ward identities** constrain certain OPE coefficients

The Ward identity fixes the TT OPE coefficient of the identity:

$$\lambda_{TT\mathbf{1}} = \sqrt{C_T} \tag{23}$$

where C_T is the coefficient in $\langle T(x)T(0)\rangle = \frac{C_T}{x^{2d}} I_{\mu\nu,\rho\sigma}$.

For the stress tensor contribution:

$$\lambda^2_{TTT} = \frac{d^2}{(d-1)^2} \cdot \frac{n_T}{C_T} \tag{24}$$

where n_T is a theory-dependent number related to a/c.

3.4 The Extremal Functional Method

The bootstrap bound is computed by finding a **linear functional** α that yields a contradiction:

Definition 3.2 (Extremal Functional)**.** A functional α acting on crossing equations typically takes the form:

$$\alpha[F] = \sum_{m,n} a_{mn} \partial_z^m \partial_{\bar{z}}^n F(u,v)\Big|_{z=\bar{z}=1/2} \tag{25}$$

evaluated at the symmetric point.

The bootstrap argument proceeds as follows:

1. **Assume:** No scalar operator exists with $\Delta < \Delta_{\text{gap}}$
2. **Find functional:** Search for α satisfying:

$$\alpha[F_{\mathbf{1}}] > 0 \quad \text{(positive on identity)} \tag{26}$$

$$\alpha[F_{\Delta,\ell}] \geq 0 \quad \text{for all allowed } (\Delta, \ell) \tag{27}$$

3. **Derive contradiction:** Apply α to crossing equation:

$$0 = \alpha\left[\sum_{\mathcal{O}} \lambda^2 F_{\mathcal{O}}\right] = \sum_{\mathcal{O}} \lambda^2 \alpha[F_{\mathcal{O}}] > 0 \tag{28}$$

   The inequality uses $\lambda^2 > 0$ (unitarity) and $\alpha[F_{\mathcal{O}}] \geq 0$.

4. **Conclusion:** No CFT can have gap larger than Δ_{gap}.

3.5 Central Charge Bounds: a/c from TTTT Bootstrap

The Hofman-Maldacena bound comes from requiring positive energy flux:

Theorem 3.2 (Hofman-Maldacena Bound)**.** In any unitary 4d CFT:

$$\boxed{\frac{1}{3} \leq \frac{a}{c} \leq \frac{31}{18}} \tag{29}$$

- Lower bound saturated by: free vector field
- Upper bound saturated by: free Weyl fermion

The TTTT bootstrap can **strengthen** this by:

- Including full crossing symmetry (not just energy positivity)
- Incorporating spin-0 and spin-4 contributions
- Adding gap assumptions on the spectrum

3.6 SDP Formulation for Bootstrap Bounds

The search for an extremal functional is formulated as a **semidefinite program**:

$$\begin{aligned} \text{Find:} \quad & \alpha = (\alpha_1, \ldots, \alpha_N) \\ \text{Subject to:} \quad & \vec{\alpha} \cdot \vec{F}_{\mathbf{1}} > 0 \\ & \vec{\alpha} \cdot \vec{F}_{\Delta,\ell} \geq 0 \quad \forall \Delta \geq \Delta_\ell^{\text{gap}}, \ \ell = 0, 2, 4, \ldots \end{aligned} \tag{30}$$

The continuous positivity constraint is handled via:

1. **Discretization:** Sample at finitely many Δ values
2. **Polynomial approximation:** Represent as polynomial in Δ and require non-negativity
3. **Sum-of-squares (SOS):** Write polynomial as sum of squares, which is an SDP constraint

Theorem 3.3 (SOS Representation)**.** A polynomial $p(\Delta) \geq 0$ on $[\Delta_{\text{gap}}, \infty)$ can be written as:

$$p(\Delta) = s_1(\Delta) + (\Delta - \Delta_{\text{gap}}) s_2(\Delta) \tag{31}$$

where s_1, s_2 are sums of squares. This reduces to a linear matrix inequality (LMI).

4 Implementation Approach

4.1 Phase 1: Conformal Block Computation (Months 1–2)

Goal: Build a high-precision calculator for conformal blocks in arbitrary dimension d.

```python
from mpmath import mp, mpf, mpc, hyp2f1
import numpy as np

mp.dps = 100  # High precision for bootstrap

class ConformalBlockCalculator:
    """
    Compute conformal blocks in arbitrary dimension d.
    """

    def __init__(self, d, precision=100):
        """
        Initialize calculator for dimension d.

        Args:
            d: Spacetime dimension
            precision: Decimal digit precision
        """
        self.d = mpf(d)
        mp.dps = precision

    def scalar_block_4d(self, Delta, ell, z, zbar):
        """
        Exact conformal block in d=4.

        G_{Delta,ell}(z,zbar) = (z*zbar)/(z-zbar) *
            [k_{Delta+ell}(z)*k_{Delta-ell-2}(zbar) - (z<->zbar)]
        """
        z, zbar = mpc(z), mpc(zbar)
        Delta, ell = mpf(Delta), mpf(ell)

        h = (Delta + ell) / 2
        hbar = (Delta - ell) / 2

        def k_beta(beta, x):
            """Hypergeometric building block."""
            return x**(beta/2) * hyp2f1(beta/2, beta/2, beta, x)

        prefactor = (z * zbar) / (z - zbar)
        term1 = k_beta(2*h, z) * k_beta(2*hbar - 2, zbar)
        term2 = k_beta(2*h, zbar) * k_beta(2*hbar - 2, z)

        return complex(prefactor * (term1 - term2))

    def block_recursion(self, Delta, ell, z, zbar, max_order=60):
        """
        Conformal block via Zamolodchikov-type recursion.
        Works for arbitrary d.

        Args:
            Delta: Scaling dimension
            ell: Spin
            z, zbar: Cross-ratio variables
            max_order: Truncation order

        Returns:
            Conformal block value
        """
        d = self.d
        Delta, ell = mpf(Delta), mpf(ell)
        z, zbar = mpc(z), mpc(zbar)

        # Radial coordinates for stability
        rho = mp.sqrt(z * zbar)
        eta = (z + zbar) / (2 * mp.sqrt(z * zbar))

        # Initialize recursion
        # G = rho^Delta * sum_{m,n} c_{m,n} rho^m C_n(eta)
        # where C_n are Gegenbauer polynomials

        result = mpf(0)

        for m in range(max_order):
            coeff_m = self._recursion_coefficient(Delta, ell, m, d)
```

```python
            result += coeff_m * rho**(Delta + m)

        return complex(result)

    def _recursion_coefficient(self, Delta, ell, m, d):
        """
        Compute recursion coefficient c_m.
        Based on Casimir eigenvalue structure.
        """
        if m == 0:
            return mpf(1)

        # Recursion from Kos-Poland-Simmons-Duffin
        c_Delta_ell = Delta * (Delta - d) + ell * (ell + d - 2)

        # Implementation of recursion relation
        # c_m = f(c_{m-1}, c_{m-2}, ...) from Casimir equation

        # Placeholder for full implementation
        return mpf(0)  # Replace with actual recursion

    def verify_casimir(self, Delta, ell, z, zbar):
        """
        Verify block satisfies Casimir equation.
        """
        d = self.d
        expected_eigenvalue = Delta * (Delta - d) + ell * (ell + d - 2)

        G = self.block_recursion(Delta, ell, z, zbar)
        DG = self._apply_casimir(Delta, ell, z, zbar)

        ratio = DG / G if G != 0 else 0

        error = abs(ratio - expected_eigenvalue)
        return error < 1e-10, error, expected_eigenvalue
```

Listing 1: Conformal block via Casimir recursion

```python
def validate_conformal_blocks():
    """
    Comprehensive validation of conformal block computation.
    """
    calc = ConformalBlockCalculator(d=4)

    print("Testing conformal blocks...")

    # Test 1: Identity block (Delta=0, ell=0) = 1
    z, zbar = 0.3, 0.2
    G_id = calc.scalar_block_4d(0, 0, z, zbar)
    assert abs(G_id - 1.0) < 1e-10, f"Identity block failed: {G_id}"
    print("  Identity block: PASSED")

    # Test 2: Casimir eigenvalue
    for Delta in [2.0, 3.5, 5.0, 7.0]:
        for ell in [0, 2, 4]:
            passed, error, expected = calc.verify_casimir(Delta, ell, z, zbar)
            assert passed, f"Casimir failed for ({Delta}, {ell}): error={error}
    "
    print("  Casimir eigenvalue: PASSED")

    # Test 3: Compare recursion vs closed form (d=4)
    for Delta in np.linspace(2, 10, 20):
        for ell in [0, 2]:
            G_exact = calc.scalar_block_4d(Delta, ell, z, zbar)
            G_rec = calc.block_recursion(Delta, ell, z, zbar)
            error = abs(G_exact - G_rec) / abs(G_exact)
            assert error < 1e-8, f"Mismatch at ({Delta}, {ell})"
    print("  Recursion vs exact: PASSED")

    # Test 4: Symmetry z <-> zbar for ell=0
    G1 = calc.scalar_block_4d(3.0, 0, 0.3, 0.2)
    G2 = calc.scalar_block_4d(3.0, 0, 0.2, 0.3)
    assert abs(G1 - G2) < 1e-10, "Symmetry failed for scalar block"
    print("  Scalar symmetry: PASSED")

    print("All conformal block tests PASSED")
    return True
```

Listing 2: Block validation suite

4.2 Phase 2: Crossing Matrix Setup (Months 2–3)

Goal: Compute derivatives of crossing vectors at the symmetric point.

```
import numpy as np
from functools import lru_cache

class CrossingVectorCalculator:
    """
    Compute crossing vectors for bootstrap.
    """

    def __init__(self, d, delta_ext=None):
        """
        Args:
            d: Spacetime dimension
            delta_ext: External operator dimension (for TTTT, this is d)
        """
        self.d = d
        self.delta_ext = delta_ext if delta_ext else d
        self.block_calc = ConformalBlockCalculator(d)

    def crossing_vector(self, Delta, ell, z, zbar):
        """
        Compute F_{Delta,ell}(z,zbar) = G_s(u,v) - G_t(v,u)

        For identical external operators.
        """
        u = z * zbar
        v = (1 - z) * (1 - zbar)

        # s-channel block
        G_s = self.block_calc.block_recursion(Delta, ell, z, zbar)

        # t-channel: swap z <-> 1-z
        z_t = 1 - z
        zbar_t = 1 - zbar
        G_t = self.block_calc.block_recursion(Delta, ell, z_t, zbar_t)

        # Crossing vector with appropriate prefactors
        F = (v ** self.delta_ext) * G_s - (u ** self.delta_ext) * G_t

        return F

    def derivative_table(self, Delta, ell, max_deriv=15):
        """
        Compute derivatives of crossing vector at z = zbar = 1/2.

        Returns:
            Dictionary mapping (m, n) -> d^{m+n}F / dz^m dzbar^n |_{z=zbar=1/2}
        """
        eps = 1e-8  # Finite difference step

        deriv_table = {}

        for m in range(max_deriv + 1):
            for n in range(max_deriv + 1 - m):
                deriv_table[(m, n)] = self._numerical_derivative(
                    Delta, ell, m, n, eps
                )

        return deriv_table

    def _numerical_derivative(self, Delta, ell, m, n, eps):
        """
        Numerical derivative using finite differences.
        """
        z0, zbar0 = 0.5, 0.5

        # High-order finite difference stencil
        stencil_coeffs = [
            -1/12, 4/3, -5/2, 4/3, -1/12
        ]  # 4th order accuracy

        result = 0.0

        for i, c_i in enumerate(stencil_coeffs):
            for j, c_j in enumerate(stencil_coeffs):
                z = z0 + (i - 2) * eps
                zbar = zbar0 + (j - 2) * eps
                F = self.crossing_vector(Delta, ell, z, zbar)
                result += c_i * c_j * F

        result /= eps**(m + n)

        return complex(result)

    def build_crossing_matrix(self, Delta_grid, ell_grid, max_deriv=15):
        """
        Build matrix of crossing vector derivatives.

        Each row: different (Delta, ell)
        Each column: different derivative (m, n)
        """
        n_ops = len(Delta_grid) * len(ell_grid)
        n_derivs = (max_deriv + 1) * (max_deriv + 2) // 2

        matrix = np.zeros((n_ops, n_derivs), dtype=complex)

        row = 0
        for Delta in Delta_grid:
            for ell in ell_grid:
                deriv_table = self.derivative_table(Delta, ell, max_deriv)

                col = 0
                for m in range(max_deriv + 1):
                    for n in range(max_deriv + 1 - m):
                        matrix[row, col] = deriv_table[(m, n)]
                        col += 1
                row += 1

        return matrix
```

Listing 3: Crossing vector computation

4.3 Phase 3: SDP Formulation for Bounds (Months 3–5)

Goal: Implement semidefinite programming solver for bootstrap.

```
import cvxpy as cp
import numpy as np

class BootstrapSDP:
    """
    Semidefinite programming formulation for conformal bootstrap.
    """

    def __init__(self, d, delta_gap, ell_max=10, deriv_order=15):
        """
        Initialize bootstrap SDP.

        Args:
            d: Spacetime dimension
            delta_gap: Assumed gap in scalar channel
            ell_max: Maximum spin to include
            deriv_order: Number of derivatives in functional
        """
        self.d = d
        self.delta_gap = delta_gap
        self.ell_max = ell_max
        self.deriv_order = deriv_order

        self.crossing_calc = CrossingVectorCalculator(d)

        # Compute number of functional components
        self.n_components = (deriv_order + 1) * (deriv_order + 2) // 2

    def setup_constraints(self, Delta_sample_points=100):
        """
        Set up SDP constraints.

        Returns:
            problem: CVXPY problem
            alpha: Functional variable
        """
        # Functional coefficients
        alpha = cp.Variable(self.n_components)

        constraints = []

        # Constraint 1: Positive on identity
        F_identity = self._crossing_vector_flat(0, 0)
        constraints.append(alpha @ F_identity >= 1)  # Normalization

        # Constraint 2: Positive on allowed operators
        for ell in range(0, self.ell_max + 1, 2):  # Even spins only

            # Unitarity bound for this spin
            if ell == 0:
                Delta_min = self.delta_gap  # Imposed gap
            else:
                Delta_min = self.d + ell - 2  # Unitarity bound
```

```python

            # Sample Delta values
            Delta_max = 30.0  # Large enough cutoff
            Delta_samples = np.linspace(Delta_min, Delta_max, Delta_sample_points)

            for Delta in Delta_samples:
                F = self._crossing_vector_flat(Delta, ell)
                constraints.append(alpha @ F >= 0)

        # Feasibility problem
        problem = cp.Problem(cp.Minimize(0), constraints)

        return problem, alpha

    def _crossing_vector_flat(self, Delta, ell):
        """
        Get crossing vector as flat array of derivatives.
        """
        deriv_table = self.crossing_calc.derivative_table(Delta, ell, self.deriv_order)

        F_flat = []
        for m in range(self.deriv_order + 1):
            for n in range(self.deriv_order + 1 - m):
                F_flat.append(deriv_table[(m, n)].real)

        return np.array(F_flat)

    def find_gap_bound(self, tolerance=0.01):
        """
        Binary search for maximum allowed gap.
        """
        Delta_min = (self.d - 2) / 2  # Unitarity bound for scalars
        Delta_max = 10.0

        print(f"Searching for gap bound in [{Delta_min:.2f}, {Delta_max:.2f}]")

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

            problem, alpha = self.setup_constraints()
            problem.solve(solver=cp.MOSEK, verbose=False)

            if problem.status == cp.OPTIMAL:
                # Feasible: functional exists, gap is EXCLUDED
                Delta_max = Delta_mid
                print(f"  Delta_gap = {Delta_mid:.4f}: EXCLUDED")
            else:
                # Infeasible: no functional, gap is ALLOWED
                Delta_min = Delta_mid
                print(f"  Delta_gap = {Delta_mid:.4f}: ALLOWED")

        bound = (Delta_min + Delta_max) / 2
        print(f"Gap bound: Delta_gap <= {bound:.4f}")

        return bound

    def extract_extremal_functional(self, delta_gap_boundary):
        """
        Extract extremal functional at the boundary.
        """
        self.delta_gap = delta_gap_boundary
        problem, alpha = self.setup_constraints()
        problem.solve(solver=cp.MOSEK, verbose=True)

        if problem.status == cp.OPTIMAL:
            return alpha.value
        return None
```

Listing 4: SDP bootstrap solver

4.4 Phase 4: Extremal Functional Method (Months 5–7)

Goal: Extract spectrum from extremal functional zeros.

```python
from scipy.optimize import brentq
import numpy as np

class ExtremalAnalyzer:
    """
    Analyze extremal functionals to extract CFT data.
    """

    def __init__(self, bootstrap_sdp, alpha_extremal):
        """
        Args:
            bootstrap_sdp: Bootstrap SDP instance
            alpha_extremal: Extremal functional coefficients
        """
        self.sdp = bootstrap_sdp
        self.alpha = alpha_extremal

    def functional_value(self, Delta, ell):
        """
        Evaluate alpha.F(Delta, ell).
        """
        F = self.sdp._crossing_vector_flat(Delta, ell)
        return self.alpha @ F

    def find_zeros(self, ell, Delta_min, Delta_max, n_search=1000):
        """
        Find zeros of extremal functional for given spin.
        """
        zeros = []

        Delta_search = np.linspace(Delta_min, Delta_max, n_search)

        for i in range(len(Delta_search) - 1):
            val1 = self.functional_value(Delta_search[i], ell)
            val2 = self.functional_value(Delta_search[i+1], ell)

            # Sign change indicates zero
            if val1 * val2 < 0:
                try:
                    Delta_zero = brentq(
                        lambda D: self.functional_value(D, ell),
                        Delta_search[i],
                        Delta_search[i+1]
                    )
                    zeros.append(Delta_zero)
                except:
                    pass

        return zeros

    def extract_spectrum(self):
        """
        Extract full spectrum from extremal functional.
        """
        spectrum = []

        for ell in range(0, self.sdp.ell_max + 1, 2):
            if ell == 0:
                Delta_min = self.sdp.delta_gap
            else:
                Delta_min = self.sdp.d + ell - 2

            zeros = self.find_zeros(ell, Delta_min, 30.0)

            for Delta in zeros:
                spectrum.append({
                    'Delta': Delta,
                    'ell': ell,
                    'type': 'double_zero' if self.is_double_zero(Delta, ell)
 else 'simple'
                })

        return spectrum

    def is_double_zero(self, Delta, ell, eps=1e-4):
        """
        Check if zero is a double zero (derivative also vanishes).
        """
        val = self.functional_value(Delta, ell)
        deriv = (self.functional_value(Delta + eps, ell) -
                 self.functional_value(Delta - eps, ell)) / (2 * eps)

        return abs(val) < 1e-8 and abs(deriv) < 1e-6

    def extract_ope_coefficients(self, spectrum):
        """
        Extract OPE coefficients from crossing equation.

        At extremal point: sum_O lambda_O^2 F_O = 0
        The functional satisfies: alpha.F_O = 0 for O in spectrum
        """
        n_ops = len(spectrum)

        # Build system of equations
        F_matrix = np.zeros((self.sdp.n_components, n_ops))

        for i, op in enumerate(spectrum):
            F_matrix[:, i] = self.sdp._crossing_vector_flat(op['Delta'], op['
 ell'])

        # Solve crossing: sum lambda^2 F = 0
        # Use SVD to find null space
        U, S, Vh = np.linalg.svd(F_matrix.T)

        # Smallest singular value gives null vector
        lambda_sq = Vh[-1]

        # Normalize: lambda^2_identity = 1
        lambda_sq = lambda_sq / lambda_sq[0]

        # Assign to spectrum
        for i, op in enumerate(spectrum):
            op['lambda_sq'] = lambda_sq[i]

        return spectrum
```

Listing 5: Extremal spectrum extraction

4.5 Phase 5: TTTT-Specific Implementation (Months 6–8)

Goal: Handle tensor structures for stress tensor correlator.

```
class TTTTBootstrap:
    """
    Bootstrap for stress tensor four-point function.
    Handles multiple tensor structures.
    """

    def __init__(self, d=4):
        self.d = d

        # Number of tensor structures in TTTT
        # d=4: 5 structures
        # d=3: 3 structures
        self.n_structures = self._count_structures(d)

    def _count_structures(self, d):
        """Count independent tensor structures for TTTT."""
        if d == 4:
            return 5
        elif d == 3:
            return 3
        else:
            # General formula from CFT literature
            return (d+2)*(d-1)//2 - 1

    def setup_crossing_system(self, Delta_gap_scalar, a_over_c_range):
        """
        Set up crossing equations for TTTT.

        Returns system of equations, one per tensor structure.
        """
        # For each structure I, crossing equation:
        # sum_O lambda^2_O G^I_O(u,v) = sum_O lambda^2_O G^I_O(v,u)

        crossing_systems = []

        for I in range(self.n_structures):
            system_I = self._crossing_for_structure(I, Delta_gap_scalar)
            crossing_systems.append(system_I)

        return crossing_systems

    def _crossing_for_structure(self, structure_index, Delta_gap):
        """
        Build crossing equation for given tensor structure.
        """
        # Placeholder: full implementation requires
        # tensor structure decomposition from literature
        # (Costa et al., Dymarsky et al.)

        pass

    def ward_identity_constraints(self, alpha):
        """
```

```python
        Implement Ward identity constraints.

        Conservation of T implies:
        - Specific relations between OPE coefficients
        - Fixes identity and T contributions
        """
        constraints = []

        # Ward identity: lambda_{TT1} fixed by C_T
        # lambda_{TTT} constrained by central charges

        return constraints

    def bound_a_over_c(self, Delta_gap_scalar=None, deriv_order=12):
        """
        Find allowed range of a/c.

        Returns:
            (a_c_min, a_c_max): Allowed range
        """
        # Set up SDP with a/c as variable
        a_c = cp.Variable()

        constraints = []

        # Add crossing constraints
        # Add unitarity constraints
        # Add Ward identity constraints

        # Minimize a/c
        prob_min = cp.Problem(cp.Minimize(a_c), constraints)
        prob_min.solve(solver=cp.MOSEK)
        a_c_min = a_c.value

        # Maximize a/c
        prob_max = cp.Problem(cp.Maximize(a_c), constraints)
        prob_max.solve(solver=cp.MOSEK)
        a_c_max = a_c.value

        return a_c_min, a_c_max
```

Listing 6: TTTT bootstrap setup

4.6 Phase 6: Certificate Generation (Months 8–10)

Goal: Generate machine-verifiable proofs of bounds.

```python
import json
import hashlib
from datetime import datetime

class BootstrapCertificate:
    """
    Machine-verifiable certificate for bootstrap bounds.
    """

    def __init__(self, bound_type, bound_value, functional, metadata):
        """
        Args:
            bound_type: 'gap_upper' or 'gap_lower' or 'ope_bound'
            bound_value: Numerical value of bound
            functional: Extremal functional coefficients
            metadata: Dictionary with d, deriv_order, etc.
        """
        self.bound_type = bound_type
        self.bound_value = bound_value
        self.functional = functional
        self.metadata = metadata
        self.timestamp = datetime.now().isoformat()

    def to_json(self, filename):
        """Export certificate to JSON."""
        cert = {
            'version': '1.0',
            'timestamp': self.timestamp,
            'bound_type': self.bound_type,
            'bound_value': float(self.bound_value),
            'functional': [float(a) for a in self.functional],
            'metadata': self.metadata,
            'verification': self._generate_verification_data(),
            'checksum': self._compute_checksum()
        }

        with open(filename, 'w') as f:
            json.dump(cert, f, indent=2)

        print(f"Certificate saved to {filename}")

    def _generate_verification_data(self):
        """Generate data for independent verification."""
        return {
            'identity_value': self._eval_on_identity(),
            'sample_positivity': self._sample_checks(n_samples=50),
            'crossing_residual': self._crossing_residual()
        }

    def _eval_on_identity(self):
        """Evaluate functional on identity."""
        # F_identity for the specific setup
        d = self.metadata['d']
        deriv_order = self.metadata['deriv_order']

        # Compute identity crossing vector
        F_id = compute_identity_vector(d, deriv_order)

        return float(np.dot(self.functional, F_id))

    def _sample_checks(self, n_samples):
        """Check positivity at sample points."""
        checks = []

        d = self.metadata['d']
        delta_gap = self.metadata.get('delta_gap', 1.0)

        for ell in [0, 2, 4]:
            delta_min = delta_gap if ell == 0 else d + ell - 2

            for Delta in np.linspace(delta_min, 20, n_samples // 3):
                F = compute_crossing_vector(Delta, ell, d, self.metadata['
    deriv_order'])
                val = float(np.dot(self.functional, F))

                checks.append({
                    'Delta': float(Delta),
                    'ell': int(ell),
                    'value': val,
```

```python
                    'positive': val >= -1e-10
                })

        return checks

    def _crossing_residual(self):
        """Compute crossing equation residual."""
        # Should be ~0 for valid certificate
        return 0.0

    def _compute_checksum(self):
        """Compute SHA256 checksum of certificate data."""
        data = json.dumps({
            'bound_value': float(self.bound_value),
            'functional': [float(a) for a in self.functional],
            'metadata': self.metadata
        }, sort_keys=True)

        return hashlib.sha256(data.encode()).hexdigest()


def verify_certificate(cert_file):
    """
    Independent verification of bootstrap certificate.

    Returns True if certificate is valid.
    """
    with open(cert_file, 'r') as f:
        cert = json.load(f)

    print(f"Verifying certificate: {cert['bound_type']} = {cert['bound_value']}
    ")

    # Check 1: Checksum
    recomputed = hashlib.sha256(json.dumps({
        'bound_value': cert['bound_value'],
        'functional': cert['functional'],
        'metadata': cert['metadata']
    }, sort_keys=True).encode()).hexdigest()

    if recomputed != cert['checksum']:
        print("FAILED: Checksum mismatch")
        return False
    print("  Checksum: PASSED")

    # Check 2: Identity positivity
    id_val = cert['verification']['identity_value']
    if id_val <= 0:
        print(f"FAILED: Identity value = {id_val} <= 0")
        return False
    print(f"  Identity positivity: PASSED (value = {id_val:.6f})")

    # Check 3: Sample positivity
    failures = [c for c in cert['verification']['sample_positivity']
                if not c['positive']]
    if failures:
        print(f"FAILED: {len(failures)} positivity violations")
        for f in failures[:5]:
            print(f"    ({f['Delta']:.3f}, {f['ell']}): {f['value']:.2e}")
        return False
    print(f"  Sample positivity: PASSED ({len(cert['verification']['
    sample_positivity'])} points)")

    print("Certificate VERIFIED")
    return True
```

Listing 7: Certificate generation

5 Research Directions

5.1 Direction 1: Gap Bounds in $d = 3, 4, 6$

Research Direction

Objective: Systematically compute maximum allowed gap to first scalar in $T \times T$ OPE across dimensions.
Method:

1. Implement conformal blocks for each d
2. Set up TTTT crossing equations with tensor structures
3. Binary search for gap bound using SDP
4. Extract extremal functional at boundary

Expected Results:

- $d = 3$: Compare with 3d Ising, $O(N)$ models
- $d = 4$: Compare with free theories, $\mathcal{N} = 4$ SYM
- $d = 6$: Constrain $(2, 0)$ theory spectrum

Key Question: Is there a universal upper bound on Δ_{gap} independent of c_T?

5.2 Direction 2: OPE Coefficient Bounds

Research Direction

Objective: Bound the OPE coefficient $\lambda^2_{TT\mathcal{O}}$ for the first scalar $\mathcal{O}$ in $T \times T$.
Approach:

1. Modify SDP to optimize over λ^2
2. Fix $(\Delta_{\mathcal{O}}, c_T)$ and find allowed range
3. Map out allowed region in $(\Delta_{\mathcal{O}}, \lambda^2)$ space

Applications:

- Test if known CFTs (free, $\mathcal{N} = 4$) lie on boundary
- Holographic interpretation: bulk cubic couplings
- Identify "extremal" theories

5.3 Direction 3: Comparison with Known CFTs

Research Direction

Objective: Locate known CFTs within bootstrap-allowed region.
Test Cases:

Theory	d	a/c	Notes
Free scalar	4	$3/1 = 3$	Minimal c
Free fermion	4	$11/6$	Near upper HM bound
Free vector	4	$31/9$	Saturates lower HM bound
$\mathcal{N} = 4$ SYM	4	1	Supersymmetric
3d Ising	3	–	Kink in bootstrap

Method:

1. Compute spectrum of each theory
2. Check if it lies on bootstrap boundary (extremal)
3. If extremal, verify by extracting spectrum from functional

5.4 Direction 4: Holographic Interpretation

Research Direction

Objective: Translate CFT bounds to bulk AdS physics.
Holographic Dictionary:

- $\Delta_{\text{gap}} \leftrightarrow$ Lightest bulk field: $m^2L^2 = \Delta(\Delta - d)$
- $c_T \leftrightarrow$ Newton constant: $G_N \sim L^{d-1}/c_T$
- $\lambda_{TT\mathcal{O}} \leftrightarrow$ Bulk cubic coupling

Questions:

1. What do CFT bounds imply for allowed bulk actions?
2. Connection to Swampland conjectures?
3. Is pure gravity (no matter) ever consistent?

Key Insight: The a/c ratio in 4d is related to higher-derivative corrections in the bulk:

$$a/c - 1 \sim \frac{\alpha'}{L^2}(\text{Gauss-Bonnet correction}) \tag{32}$$

5.5 Direction 5: Improved a/c Bounds

Research Direction

Objective: Use full TTTT bootstrap to improve Hofman-Maldacena bounds.
Current Bounds:

$$\frac{1}{3} \le \frac{a}{c} \le \frac{31}{18} \tag{33}$$

Bootstrap Improvements:

1. Include full crossing (not just energy positivity)
2. Add gap assumptions on scalar spectrum
3. Use mixed correlator $(\phi\phi TT)$

Target: Determine if bounds tighten for CFTs with:

- No relevant scalars
- Supersymmetry
- Large c_T (holographic)

5.6 Direction 6: Mixed Correlator Bootstrap

Research Direction

Objective: Include $\langle\phi\phi TT\rangle$ for stronger constraints.
Setup: For a CFT with scalar ϕ (dimension Δ_ϕ) and stress tensor T:

- $\langle\phi\phi\phi\phi\rangle$: Standard scalar bootstrap
- $\langle\phi\phi TT\rangle$: Mixed correlator
- $\langle TTTT\rangle$: Stress tensor bootstrap

Advantage: The combined system is more overdetermined, leading to tighter bounds or isolated theories.
Challenge: Computational complexity increases due to:

- Multiple tensor structures
- Cross-channel constraints
- Larger SDP problem

6 Success Criteria

6.1 Minimum Viable Result (4–5 months)

✓ Conformal block calculator working in $d = 3, 4$ with 50+ digit precision

✓ Crossing equations for scalar four-point function implemented

✓ Basic SDP solver producing gap bounds

✓ Reproduction of known results: 3d Ising kink location

✓ **One new rigorous bound** on scalar gap or OPE coefficient

6.2 Strong Result (6–8 months)

✓ TTTT bootstrap implemented with all tensor structures

✓ a/c bounds computed and compared with Hofman-Maldacena

✓ Extremal functional extraction working

✓ Spectrum extraction at boundary points

✓ Results for $d = 3, 4, 6$

✓ All bounds in machine-verifiable certificate form

6.3 Publication-Quality Result (10–12 months)

✓ Mixed correlator system ($\phi\phi TT + TTTT$)

✓ Novel a/c bounds with gap assumptions

✓ Comparison with $\mathcal{N} = 4$ SYM predictions

✓ Holographic interpretation documented

✓ Public code repository with documentation

✓ ArXiv preprint with verifiable certificates

7 Verification Protocol

7.1 Conformal Block Verification

```python
def verify_blocks_comprehensive(d):
    """
    Full test suite for conformal blocks in dimension d.
    """
    calc = ConformalBlockCalculator(d)
    tests = {'passed': 0, 'failed': 0}

    # Test 1: Identity block = 1
    z, zbar = 0.3 + 0.01j, 0.2 - 0.01j
    G_id = calc.block_recursion(0, 0, z, zbar)
    if abs(G_id - 1.0) < 1e-8:
        tests['passed'] += 1
    else:
        tests['failed'] += 1
        print(f"Identity block FAILED: {G_id}")

    # Test 2: Casimir eigenvalue
    for Delta, ell in [(3.0, 0), (4.5, 2), (6.0, 4)]:
        passed, error, expected = calc.verify_casimir(Delta, ell, z, zbar)
        if passed:
            tests['passed'] += 1
        else:
            tests['failed'] += 1
            print(f"Casimir FAILED at ({Delta}, {ell}): error = {error}")

    # Test 3: Boundary behavior
    # As z -> 0: G ~ z^{Delta/2} for ell = 0
    z_small = 0.001
    G = calc.block_recursion(4.0, 0, z_small, z_small)
    expected_scaling = z_small ** 2.0  # Delta/2 = 2
    if abs(G / expected_scaling - 1) < 0.1:  # 10% error at small z
        tests['passed'] += 1
    else:
        tests['failed'] += 1

    # Test 4: Symmetry
    G1 = calc.block_recursion(3.0, 0, 0.3, 0.2)
    G2 = calc.block_recursion(3.0, 0, 0.2, 0.3)
    if abs(G1 - G2) < 1e-8:
        tests['passed'] += 1
    else:
        tests['failed'] += 1

    print(f"Block tests: {tests['passed']} passed, {tests['failed']} failed")
    return tests['failed'] == 0
```

Listing 8: Comprehensive block tests

7.2 SDP Solution Verification

```python
def verify_sdp_solution(alpha, sdp_instance, n_fine=5000):
    """
    Verify extremal functional satisfies all constraints.
    """
    results = {
        'identity_positive': False,
        'all_positive': True,
        'min_value': float('inf'),
        'min_location': None
    }

    # Check identity
    F_id = sdp_instance._crossing_vector_flat(0, 0)
    id_val = np.dot(alpha, F_id)
    results['identity_positive'] = (id_val > 0)
    print(f"Identity: alpha.F = {id_val:.6f}")

    # Fine-grained positivity check
    for ell in range(0, sdp_instance.ell_max + 1, 2):
        Delta_min = sdp_instance.delta_gap if ell == 0 else sdp_instance.d +
    ell - 2

        for Delta in np.linspace(Delta_min, 30, n_fine):
            F = sdp_instance._crossing_vector_flat(Delta, ell)
            val = np.dot(alpha, F)

            if val < results['min_value']:
                results['min_value'] = val
                results['min_location'] = (Delta, ell)

            if val < -1e-8:
                results['all_positive'] = False
                print(f"Violation at ({Delta:.4f}, {ell}): {val:.2e}")

    print(f"Minimum value: {results['min_value']:.2e} at {results['min_location
    ']}")

    results['verified'] = results['identity_positive'] and results['all_positive']

    return results
```
Listing 9: SDP verification

7.3 Literature Comparison

```
def compare_with_literature():
    """
    Compare computed bounds with published bootstrap results.
    """
    # Known 3d Ising data (Kos-Poland-Simmons-Duffin 2014)
    ising_3d = {
        'Delta_sigma': 0.5181489,
        'Delta_epsilon': 1.412625,
        'C_sigma_sigma_epsilon': 1.0518537
    }

    # 3d O(N) data
    on_3d = {
        2: {'Delta_phi': 0.5190, 'Delta_s': 1.51},
        3: {'Delta_phi': 0.5190, 'Delta_s': 1.59}
    }

    # Hofman-Maldacena bounds
    hm_bounds = {
        'a_over_c_min': 1/3,
        'a_over_c_max': 31/18
    }

    print("Literature comparison:")
    print(f"  3d Ising: Delta_sigma = {ising_3d['Delta_sigma']}")
    print(f"  HM bounds: {hm_bounds['a_over_c_min']:.4f} <= a/c <= {hm_bounds['a_over_c_max']:.4f}")

    # Compare with our computed bounds
    # our_gap_bound = ...
    # print(f"  Our gap bound: {our_gap_bound}")
```
Listing 10: Compare with published results

8 Common Pitfalls and Mitigations

8.1 Numerical Precision in Conformal Blocks

Critical Consideration

Problem: Conformal blocks involve hypergeometric functions that can lose precision for certain parameter ranges.
Symptoms:

- NaN or Inf for large Δ
- Loss of precision near $z = 1$
- Casimir check failing

Solutions:

- Use `mpmath` with precision ≥ 100 digits
- Implement stable recursion (radial coordinates)
- Use analytic continuation formulas for $z \to 1$

8.2 SDP Solver Issues

Critical Consideration

Problem: SDP solvers can return incorrect status or fail to converge for bootstrap problems.
Symptoms:

- Different solvers give different bounds
- Bound changes with derivative order unexpectedly
- "Numerical issues" warnings

Solutions:

- Use multiple solvers: MOSEK, SDPA, SCS
- Increase precision of constraint matrices
- Verify solution by explicit evaluation
- Use SDPB for highest accuracy

8.3 Discretization Artifacts

Critical Consideration

Problem: Discretizing continuous positivity misses violations between sample points.
Symptoms:

- Bound improves with coarser grid (incorrect!)
- Functional negative at non-sampled points

Solutions:

- Use sum-of-squares (SOS) for continuous positivity
- Check functional on very fine grid post-solution
- Solve at multiple grid sizes and extrapolate

8.4 Tensor Structure Errors

Critical Consideration

Problem: TTTT has multiple tensor structures with complex index contractions. Errors propagate to wrong bounds.
Symptoms:

- a/c bounds outside Hofman-Maldacena range
- Free theory fails constraints
- Ward identity violations

Solutions:

- Verify against free field theory (exactly solvable)
- Check Ward identities explicitly
- Compare with published expressions
- Use automated tensor algebra

8.5 Convergence of Derivative Expansion

Critical Consideration

Problem: Derivative expansion may not converge uniformly.
Symptoms:

- Bound not stabilizing with derivative order
- Large oscillations in extremal functional

Solutions:

- Plot bound vs derivative order systematically
- Use radial expansion for better convergence
- Employ Pade approximants

9 Resources and References

9.1 Essential Papers

1. **Rattazzi, Rychkov, Tonni, Vichi (2008):** "Bounding scalar operator dimensions in 4D CFT" [arXiv:0807.0004]
   - Founding paper of modern numerical bootstrap
2. **El-Showk et al. (2012):** "Solving the 3D Ising Model with the Conformal Bootstrap" [arXiv:1203.6064]
   - Precision determination of 3d Ising exponents
3. **Hofman, Maldacena (2008):** "Conformal collider physics" [arXiv:0803.1467]
   - Original a/c bounds from energy positivity
4. **Dymarsky et al. (2017):** "The 3d Stress-Tensor Bootstrap" [arXiv:1708.05718]
   - TTTT bootstrap in 3d
5. **Poland, Rychkov, Vichi (2019):** "The Conformal Bootstrap: Theory, Numerical Techniques, and Applications" [arXiv:1805.04405]
   - Comprehensive review
6. **Simmons-Duffin (2017):** "The Conformal Bootstrap" (TASI lectures) [arXiv:1602.07982]
   - Pedagogical introduction

9.2 Code Libraries

- **SDPB:** High-precision SDP solver for bootstrap
  - `https://github.com/davidsd/sdpb`
  - Optimized for bootstrap, arbitrary precision
- **blocks_3d:** Conformal blocks in 3d

  - `https://gitlab.com/bootstrapcollaboration/blocks_3d`
- **mpmath:** Arbitrary precision Python
  - `pip install mpmath`
- **CVXPY:** Convex optimization
  - `pip install cvxpy`
- **MOSEK:** Commercial SDP solver (free academic license)
  - `https://www.mosek.com`

9.3 Mathematical Background

- **CFT:** Di Francesco et al., "Conformal Field Theory"
- **Representation Theory:** Fulton-Harris, "Representation Theory"
- **SDP:** Boyd-Vandenberghe, "Convex Optimization"
- **Special Functions:** DLMF (`https://dlmf.nist.gov`)

10 Milestone Checklist

10.1 Infrastructure (Months 1–2)

- ☐ Conformal block calculator for $d = 3, 4, 6$
- ☐ Casimir eigenvalue verified
- ☐ Recursion and closed-form agree ($d = 4$)
- ☐ Crossing vector computation working
- ☐ Derivative table generation
- ☐ Basic SDP solver running

10.2 Scalar Bootstrap (Months 2–3)

- ☐ Gap bound for 3d scalars reproduced
- ☐ 3d Ising "kink" identified
- ☐ Free scalar verified
- ☐ Bounds stable vs derivative order
- ☐ Multiple solvers agree

10.3 OPE Bounds (Months 3–4)

- ☐ SDP extended for OPE bounds
- ☐ Allowed region in (Δ, λ^2) mapped
- ☐ 3d Ising OPE coefficient verified

10.4 TTTT Bootstrap (Months 4–6)

- ☐ Tensor structures implemented
- ☐ Ward identities incorporated
- ☐ Spinning conformal blocks
- ☐ Selection rules verified
- ☐ Free theory passes

10.5 a/c Bounds (Months 6–8)

- ☐ Hofman-Maldacena reproduced
- ☐ Gap assumptions added
- ☐ Bounds vs gap studied
- ☐ Comparison with $\mathcal{N} = 4$ SYM

10.6 Extremal Analysis (Months 8–10)

- ☐ Extremal functional extraction
- ☐ Double zeros identified
- ☐ OPE coefficients extracted
- ☐ Extremal spectrum characterized

10.7 Publication (Months 10–12)

- ☐ All bounds have certificates
- ☐ Independent verification
- ☐ Code repository public
- ☐ Documentation complete
- ☐ ArXiv preprint submitted

11 Conclusion

The conformal bootstrap in higher dimensions provides one of the most powerful non-perturbative approaches to quantum field theory. By exploiting only fundamental axioms—conformal symmetry, unitarity, and crossing—it produces rigorous bounds applicable to all CFTs, including strongly coupled theories.

This challenge focuses on the **TTTT bootstrap**, which constrains universal CFT data: central charges and stress tensor OPE coefficients. Key outcomes include:

1. **Gap bounds:** Maximum gap to first scalar in $T \times T$ OPE
2. a/c **bounds:** Sharpening Hofman-Maldacena using full crossing
3. **Extremal theories:** Identifying CFTs saturating bootstrap bounds

4. **Holographic implications:** Constraints on bulk AdS effective actions

Physical Insight

Broader Impact:
Success in this challenge advances our understanding of:

- **Critical phenomena:** Universal features of phase transitions
- **Strongly coupled QFT:** Non-perturbative physics
- **Quantum gravity:** Via AdS/CFT, CFT bounds constrain consistent gravities
- **Mathematical physics:** Rigorous, machine-verifiable results

The development of machine-verifiable certificates ensures all results can be independently checked, establishing new standards for rigor in theoretical physics.

A Conformal Block Technical Details

A.1 Zamolodchikov Recursion

The conformal block can be expanded as:

$$G_{\Delta,\ell}(r,\eta) = r^{\Delta}\sum_{n=0}^{\infty} r^n g_n^{(\Delta,\ell)}(\eta) \tag{34}$$

where $r = |z|^{1/2}|1-z|^{1/2}$ and $\eta = (z+\bar{z})/(2\sqrt{z\bar{z}})$.
The coefficients satisfy:

$$g_n(\eta) = \sum_{m=0}^{n} A_{nm}(\Delta,\ell)C_m^{(\nu)}(\eta) \tag{35}$$

where $C_m^{(\nu)}$ are Gegenbauer polynomials with $\nu = (d-2)/2$.

A.2 Casimir Operator

In radial coordinates, the Casimir takes the form:

$$\mathcal{D} = (1-r^2)r\partial_r + (1-\eta^2)\partial_\eta^2 - (d-1)\eta\partial_\eta + \dots \tag{36}$$

The eigenvalue equation $\mathcal{D}G = c_{\Delta,\ell}G$ with:

$$c_{\Delta,\ell} = \Delta(\Delta-d) + \ell(\ell+d-2) \tag{37}$$

uniquely determines the block given boundary conditions.

B Tensor Structures for TTTT

B.1 Embedding Space

In embedding space $\mathbb{R}^{d+1,1}$, points are represented as null rays:

$$P^M = (1, x^2, x^\mu), \quad P^2 = 0 \tag{38}$$

The stress tensor lifts to:

$$T^{MN}(P) = \frac{\partial P^M}{\partial x^\mu}\frac{\partial P^N}{\partial x^\nu}T^{\mu\nu}(x) \tag{39}$$

B.2 Building Blocks

Independent tensor structures are built from:

$$H_{ij} = -2(P_i \cdot P_j)(Z_i \cdot Z_j) + 2(Z_i \cdot P_j)(Z_j \cdot P_i) \tag{40}$$

$$V_{i,jk} = \frac{(Z_i \cdot P_j)(P_i \cdot P_k) - (Z_i \cdot P_k)(P_i \cdot P_j)}{P_j \cdot P_k} \tag{41}$$

where Z_i are polarization vectors.

B.3 TTTT Structures in 4d

The five independent structures can be chosen as:

$$t_1 = H_{12}^2 H_{34}^2 \tag{42}$$

$$t_2 = H_{12} H_{34} V_{1,23} V_{2,13} V_{3,24} V_{4,14} \tag{43}$$

$$t_3 = H_{13} H_{24} V_{1,23} V_{2,13} V_{3,24} V_{4,14} \tag{44}$$

$$t_4 = V_{1,23}^2 V_{2,13}^2 V_{3,24}^2 V_{4,14}^2 \tag{45}$$

$$t_5 = \text{(cyclic combinations)} \tag{46}$$

C SDP Implementation Details

C.1 Polynomial Positivity via SOS

A polynomial $p(\Delta)$ is non-negative on $[\Delta_0, \infty)$ iff:

$$p(\Delta) = s_1(\Delta) + (\Delta - \Delta_0) s_2(\Delta) \tag{47}$$

where s_1, s_2 are sums of squares.

An SOS polynomial of degree $2d$ can be written:

$$s(\Delta) = \vec{v}(\Delta)^T Q \vec{v}(\Delta), \quad Q \succeq 0 \tag{48}$$

where $\vec{v}(\Delta) = (1, \Delta, \Delta^2, \ldots, \Delta^d)^T$.

The constraint $Q \succeq 0$ (positive semidefinite) is an LMI, making the overall problem an SDP.