PRD 04: Modular-Lightcone Bootstrap for Holographic CFTs

Pure Thought AI Challenges

January 19, 2026

Contents

Domain: Quantum Gravity & Particle Physics **Difficulty:** High **Timeline:** 7-10 months **Prerequisites:** Conformal field theory, AdS/CFT, modular forms, semidefinite programming, representation theory

—

0.1 1. Problem Statement

0.1.1 Scientific Context

The **conformal bootstrap** program, pioneered by Ferrara, Gliozzi, and Scherk (1973) and revived by Rattazzi et al. (2008), uses crossing symmetry and unitarity to constrain conformal field theories (CFTs) non-perturbatively. When combined with the **AdS/CFT correspondence** (Maldacena 1997), the bootstrap becomes a powerful tool for deriving universal properties of quantum gravity.

Large-c holographic CFTs are dual to Einstein gravity in Anti-de Sitter (AdS) space at leading order in 1/c, where c is the central charge (proportional to N^2 in AdS/CFT gauge theory duality). The **sparse spectrum** assumption — that the lightest non-conserved operator appears only at dimension 1 — corresponds to the bulk having few light states beyond the graviton, characteristic of Einstein gravity without large higher-derivative corrections.

The **gravitational bootstrap** program asks: **What are the universal constraints on large-c, sparse-spectrum CFTs imposed by consistency alone?** Key inputs:

1. **Crossing Symmetry**: The OPE decomposition of 4-point functions must be consistent in different channels $$\langle\mathcal{O}\_1(x\_1)\mathcal{O}\_2(x\_2)\mathcal{O}\_3(x\_3)\mathcal{O}\_4(x\_4)\rangle_{s\text{-channel}} = \langle\cdots\rangle_{t\text{-channel}}$$

2. **Modular Invariance**: Thermal correlators on S^{d-1}$S^1 mustrespectSL(2,)modulartransformations4/$

3. **Causality (Chaos Bound)**: The quantum Lyapunov exponent satisfies _L 2/ (Maldacena-Shenker-Stanford bound), which translates to **Regge behavior** constraints on OPE data in the s→ limit of the crossing equation

4. **Lightcone Limits**: As cross-ratios z → 0 (lightcone), the 4-point function is dominated by operators with minimal twist = - J (= scaling dimension, J = spin)

Higher-Spin Gravity: If a large-c CFT contains **higher-spin conserved currents** (spin J ¿ 2 beyond the stress tensor), the bulk dual is a higher-spin gravity theory (Vasiliev theory). The bootstrap can **rule out** such theories in certain parameter regions, proving that Einstein gravity is "rigid" — the unique consistent theory of massless spin-2 under holographic constraints.

Key Results (from literature as benchmarks): - **3D CFT**: Universal bound _gap (d-2)/2 = 0.5 for spin-0 gap (Heemskerk et al. 2009) - **Spin-4 current**: If spin-4 current exists, OPE coefficient bounded by c (Afkhami-Jeddi et al. 2019) - **Average null energy**: Constrains 1/c corrections to gravitational dynamics (Hartman et al. 2016)

0.1.2 Core Question

What are the sharp, universal bounds on: 1. **Twist gap** _gap - J_max to the first non-conserved operator (excluding identity, stress tensor, higher-spin currents) 2. **OPE coefficients** of higher-spin conserved currents (if present) 3. **1/c corrections** to gravitational coupling constants in the bulk

Derived using ONLY: - Crossing symmetry + unitarity - Modular invariance (for thermal states) - Causality/chaos bound (Regge constraints) - Large-c limit (1/c expansion)

No input from bulk gravity or string theory until verification!

0.1.3 Why This Matters

- **Quantum Gravity Consistency**: Bootstrap bounds are **non-perturbative constraints** on any UV completion of Einstein gravity (string theory, loop quantum gravity, etc.) - **Swampland Program**: Certain parameter regions are **forbidden** by consistency, defining the "landscape" vs "swampland" of effective field theories - **Universality**: Bounds apply to **all** holographic CFTs (supersymmetric or not, with or without extra symmetries), revealing universal gravitational physics - **Testable Predictions**: Compare to known AdS/CFT pairs (AdS×S, M-theory on AdS×S, etc.) to validate bootstrap assumptions

0.1.4 Pure Thought Advantages

1. **Exact non-perturbative results**: Bootstrap bounds hold for arbitrarily large interactions, unlike perturbative quantum field theory 2. **No bulk calculations needed**: Derive gravitational constraints purely from CFT consistency (boundary observables) 3. **Certificates**: Extremal functionals provide **machine-checkable proofs** that certain CFTs cannot exist 4. **Scalable**: Semidefinite programming (SDP) solvers handle 10-10 variables; parallelizable on GPUs

—

0.2 2. Mathematical Formulation

0.2.1 Conformal Block Decomposition

Consider the **stress tensor 4-point function** in a d-dimensional CFT: \$\$G(z,) $= \langle T(x\_1)T(x\_2)T(x\_3)T(x\_4)\rangle$

where \$(z,)\$ are **conformal cross-ratios**: \$\$z $= \mathrm{x}_{12}^2 x_{34}^2 \frac{}{x_{13}^2 x_{24}^2, \quad (1-z)(1-\bar{z}) = \frac{x_{14}^2 x_{23}^2}{x_{13}^2 x_{24}^2}}$ \$\$

OPE decomposition in the s-channel ($12 \to 34$): \$\$G(z,) $= \sum_{\mathcal{O}} \lambda_{TT\mathcal{O}}^2 \, g_{\Delta,J}^{(s)}(z, \bar{z})$\$\$

where: - \$g$_{\Delta,J}^{(s)}(z,\bar{z})$\$$isthe{*}{*}conformalblock{*}{*}forexchangeofoperator$\$$\mathcal{O}$\$$withdimensionandspinJ-$ \$$\lambda_{TT\mathcal{O}}$\$$isthe **OPEcoefficient** (3 - pointfunctionstructureconstant) - Sumrunsover **primaryoperators**$ \$$\mathcal{O}$\$$appearingin$\$$T \times T$\$$OPE$

Conserved currents: Stress tensor (spin-2), possible higher-spin currents (spin J = 4, 6, 8, ...) \$\$$\nabla^\mu J_{\mu\mu_2\cdots\mu_J} = 0 \quad \Rightarrow \quad \Delta_J = J + d - 2$\$\$

Non-conserved operators: Generic operators with d/2 (unitarity bound)

0.2.2 Crossing Symmetry

Crossing equation: The same 4-point function decomposed in different OPE channels must agree: \$\$$\sum_{\mathcal{O}} \lambda_{TT\mathcal{O}}^2 \, g_{\Delta,J}^{(s)}(z,\bar{z}) = \sum_{\mathcal{O}'} \lambda_{TT\mathcal{O}'}^2 \, g_{\Delta',J'}^{(t)}(v,u)$\$\$

where \$(u, v) = (z, (1-z)(1-))\$ are related cross-ratios.

Crossing vector: Define \$\$$_{\Delta,J}(z,\bar{z}) = g_{\Delta,J}^{(s)}(z,\bar{z}) - F_{d,J} \, g_{\Delta,J}^{(t)}(v,u)$\$\$

where \$F$_{d,J}$\$$isaspin-dependentfactorfromstresstensorstructure.Then:$ \$\$ $\sum_{\mathcal{O}} \lambda_{TT\mathcal{O}}^2 \, \vec{V}_{\Delta,J}(z,\bar{z}) =$ 0\$\$

This must hold for **all** \$(z,)\$ — infinitely many constraints on finitely many OPE coefficients!

0.2.3 Lightcone Limit and Twist Spectrum

Lightcone limit: \$z $\to$ 0\$$with$\$$\bar{z}$\$$fixed.Theconformalblockbehavesas:$ \$\$$g_{\Delta,J}(z,\bar{z}) \sim z^{\tau/2}\bar{z}^{\bar{\tau}/2} \quad \text{where} \quad \tau = \Delta - J$ (twist)\$\$

Leading twist: Operators with smallest \$$\tau$\$$dominatethelightcone.Forconservedcurrents:$ \$\$$\tau_J = (J + d - 2) - J = d - 2 \quad$ (universal!)\$\$

Twist gap assumption: Assume no **non-conserved** operators with \$$\tau < \tau_{\text{gap}}$\$$(exceptidentitywith$\$$\tau$ 0\$)$.Thisdefinesa **sparsespectrum**:$ \$\$Spectrum $= \{\not\Vdash, T_{\mu\nu}, J\_J, \ldots\} \cup \{\mathcal{O} : \tau_{\mathcal{O}} \geq \tau_{\text{gap}}\}$\$\$

0.2.4 Chaos Bound and Regge Constraints

Quantum chaos bound (Maldacena-Shenker-Stanford 2016): $$$\lambda\_L \leq \frac{2\pi}{\beta} \quad \text{(Lyapunov exponent at tempera}$ $1/\beta)$$$

This bound is **saturated by Einstein gravity**. Violations would indicate faster scrambling, inconsistent with causality in the bulk.

Regge limit: $s, t $\to \infty$ $with$ $s/t$ $fixed.Thecrossingequationimposes$: $$$\mathcal{G}(s,t) \sim s^{j(t)} \quad \text{where } j(t) = 1 + \frac{t}{2\pi T\_H} + O(t^2)$$$

is the **Regge trajectory**. The chaos bound constrains the intercept $j(0)$ and slope $j'(0)$.

Implementation: Impose positivity of **averaged null energy** (ANEC) operator, which encodes causality.

0.2.5 Large-c Expansion

At large central charge $c $\to \infty$$: $$$\langle TTTT \rangle = \langle TT \rangle \langle TT \rangle + \frac{1}{c}(\text{connected part}) + O(1/c^2)$$$

Graviton exchange (identity operator) dominates at leading order. The $1/c$ corrections encode: - Multi-graviton states - Stringy/higher-derivative corrections

Goal: Bound these corrections using bootstrap!

0.2.6 Certificate Specification

A **valid bootstrap certificate** proving an upper bound $$\Delta_{\text{gap}} \leq \Delta_{\max}$ $mustinclude$:

1. **Extremal Functional** $$\alpha(\Delta, J)$ $: -Definedonagrid$ (Δ_i, J_k) $withvalues$ $\alpha_{ik} \in \mathbb{R}$ $-**Positivity**: $ $\alpha(\Delta, J) \geq 0$ $forall$ $\Delta \geq \Delta_{\max}$ $andallallowedspins$ $J$ $-**Crossingviolation**: Applying$ α $tocrossingequationgivesnegativeresult$ $$\sum_{i,k} \alpha_{ik} \vec{V}_{\Delta_i, J_k}(z_0, \bar{z}_0) < 0$$ $foratleastonepoint$

2. **Numerical Precision**: - Grid spacing: $$\Delta\Delta < 0.01$, $$\Delta J = 2$ $-SDPtolerance: dualgap < 10^{-8} - Blockevaluation: 100 - digitprecisionwith`mpmath`$

3. **Verification**: - Independent code checks positivity on denser grid - Crossing violation confirmed at 10 test points - Comparison to known CFTs (e.g., 3D Ising at c = 0.95) that saturate the bound

4. **Export Format**: JSON with: - Grid points $$\{(\Delta_i, J_k)\}$ $-Functionalvalues$ $\{\alpha_{ik}\}$ $(exactrationals$ $Crossingvectorsevaluatedattestpoints - Metadata: dimensiond, centralchargec, imposedconstraints$

—

0.3 3. Implementation Approach

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

Goal: Compute $$g^{TT}_{\Delta,J}(z,\bar{z})$ $forstresstensor4-pointfunctionind = 3, 4.$

```
import numpy as np
from mpmath import mp, mpf, hyp2f1
mp.dps = 100  # 100-digit precision

def conformal_block_scalar_exchange(Delta, ell, z, zbar, d=3):
    """
    Scalar (ell=0) conformal block in d dimensions.

    Uses hypergeometric function 2F1.
    """
    # Eigenvalue of quadratic Casimir
    C_Delta_ell = Delta * (Delta - d) + ell * (ell + d - 2)

    # For ell=0 (scalar), closed form:
    # g_{Delta,0}(z,zbar) ~ z^{Delta/2} F(Delta/2, Delta/2; Delta; z)
    z_mp = mpf(z)
    z_bar_mp = mpf(zbar)

    # Split into (z, zbar) as (holomorphic, antiholomorphic)
    a = Delta / 2
    b = Delta / 2
    c = Delta

    F_z = hyp2f1(a, b, c, z_mp)
    F_zbar = hyp2f1(a, b, c, z_bar_mp)

    block = (z_mp ** (Delta/2)) * F_z * (z_bar_mp ** (Delta/2)) * F_zbar

    return float(block)


def conformal_block_spinning(Delta, J, z, zbar, d=3):
    """
    Spinning conformal block (J > 0) via recursion.

    For large J, use recursion relation or Casimir differential equation
       :
    D g_{Delta,J} = C_{Delta,J} g_{Delta,J}

    Reference: Dolan & Osborn (2011), Kos et al. (2014)
    """
    if J == 0:
        return conformal_block_scalar_exchange(Delta, 0, z, zbar, d)

    # Use recursion from J=0 case (simplified for illustration)
    # Full implementation requires Casimir equation numerical solve

    # Placeholder: linear approximation (not accurate!)
    block_J0 = conformal_block_scalar_exchange(Delta, 0, z, zbar, d)
    correction = J * (Delta - d/2) * (z - zbar) ** J / np.math.factorial
       (J)

    return block_J0 + correction


def stress_tensor_4pt_block(Delta_ex, J_ex, z, zbar, d=3):
    """
    Conformal block for stress tensor 4-point function.

      T   T T   T   block depends on external operator structure.
    Use embedding space formalism or projectors.

    Simplified: assume external operators are scalars for now.
    Full version requires tensor structure projectors.
    """
    # For stress tensor external operators, multiply by kinematic
       prefactor
    prefactor = 1.0  # Simplified; full version has (z*zbar)^d
       dependence

    return prefactor * conformal_block_spinning(Delta_ex, J_ex, z, zbar,
        d)
```

```
# Test: verify identity block
def test_identity_block():
    """Identity operator (Delta=0, J=0) should give 1."""
    z, zbar = 0.3, 0.7
    block = stress_tensor_4pt_block(0, 0, z, zbar, d=3)
    assert abs(block - 1.0) < 1e-10, f"Identity block = {block},
        expected 1"
    print("    Identity block verified")


# Test: stress tensor block
def test_stress_tensor_block():
    """Stress tensor (Delta=d, J=2) appears in T T OPE."""
    d = 3
    Delta_T = d  # Stress tensor dimension
    J_T = 2

    z, zbar = 0.2, 0.8
    block = stress_tensor_4pt_block(Delta_T, J_T, z, zbar, d)

    print(f"Stress tensor block at (z={z}, zbar={zbar}): {block:.6f}")
    assert block > 0, "Block should be positive"
    print("    Stress tensor block computed")
```

Output: Grid of conformal blocks for $(
Delta, J)$ on $[0, 10]
times
0, 2, 4, 6, 8
$.

—

0.3.2 Phase 2: Crossing Symmetry and Functional Method (Months 2-4)

Goal: Formulate crossing equation as linear system; implement functional method.

```
def crossing_vector(Delta, J, z_points, zbar_points, d=3):
    """
    Compute crossing kernel vector V_{Delta,J}(z, zbar).

    V = g^{(s)}(z, zbar) - F_{d,J} * g^{(t)}(v, u)

    where (u,v) = (z*zbar, (1-z)*(1-zbar))
    """
    vectors = []

    for z, zbar in zip(z_points, zbar_points):
        # S-channel
        g_s = stress_tensor_4pt_block(Delta, J, z, zbar, d)

        # T-channel: swap cross-ratios
        u = z * zbar
        v = (1 - z) * (1 - zbar)

        # Reconstruct z', zbar' from (u, v)
        # u = z' zbar', v = (1-z')(1-zbar')
        # Solve: z', zbar' = ...
        # Simplified: use symmetry factor
        F_dJ = compute_symmetry_factor(d, J)  # Depends on stress tensor
            structure
        g_t = stress_tensor_4pt_block(Delta, J, v, u, d)

        V = g_s - F_dJ * g_t
        vectors.append(V)

    return np.array(vectors)


def compute_symmetry_factor(d, J):
    """
    Symmetry factor relating s- and t-channel blocks.

    For stress tensor 4-point function, this is non-trivial.
    Simplified: use d-dependence only.
    """
    return (-1) ** J / (d - 1)  # Placeholder


def setup_crossing_matrix(Delta_grid, J_grid, z_points, zbar_points, d
   =3):
    """
    Build crossing matrix: each row is V_{Delta,J} at one (z, zbar)
       point.
    """
    num_points = len(z_points)
    num_ops = len(Delta_grid) * len(J_grid)

    crossing_matrix = np.zeros((num_points, num_ops))

    idx = 0
    for Delta in Delta_grid:
        for J in J_grid:
            V = crossing_vector(Delta, J, z_points, zbar_points, d)
            crossing_matrix[:, idx] = V
            idx += 1

    return crossing_matrix


def verify_crossing_known_CFT(ope_coeffs, Delta_spectrum, J_spectrum,
   z_points, zbar_points):
    """
    Verify crossing for a known CFT (e.g., generalized free field).

          ^2_{TT  ,J} V_{  ,J} should be      0.
    """
    crossing_mat = setup_crossing_matrix(Delta_spectrum, J_spectrum,
       z_points, zbar_points)

    residual = crossing_mat @ ope_coeffs

    max_violation = np.max(np.abs(residual))
    print(f"Crossing violation: {max_violation:.2e}")
    assert max_violation < 1e-8, "Crossing not satisfied!"
    print("    Crossing symmetry verified")


# Test case: Generalized Free Field (GFF)
def test_gff_crossing():
    """
    Generalized free field: T  T  OPE contains only double-trace
       operators.

     _{n,J} = 2  _   + 2n + J
      ^2 ~ 1/c (suppressed at large c)
    """
    c = 100
    Delta_phi = 1.0  # Scalar dimension

    # Double-trace spectrum
    Delta_spectrum = [2*Delta_phi + J for J in [0, 2, 4]]
    J_spectrum = [0, 2, 4]

    # OPE coefficients (1/c suppressed)
    ope_coeffs = np.array([1/c, 0.5/c, 0.3/c] * 3)

    z_points = np.linspace(0.1, 0.9, 10)
    zbar_points = np.linspace(0.1, 0.9, 10)

    verify_crossing_known_CFT(ope_coeffs, Delta_spectrum, J_spectrum,
       z_points, zbar_points)
```

—

0.3.3 Phase 3: Lightcone Limits and Twist Spectrum (Months 4-5)

Goal: Extract leading twist operators in $z
to 0$ limit.

```
def lightcone_expansion(correlator_func, zbar_fixed=0.5, z_values=None):
    """
    Expand correlator in lightcone limit z     0.

    G(z, zbar) ~   _ {  } z^{  /2} F_  (zbar)

    Extract leading twist   _min .
    """
    if z_values is None:
        z_values = [10**(-k) for k in range(1, 7)]  # z = 0.1, 0.01,
           ..., 10^-6

    log_G = []
    log_z = []

    for z in z_values:
        G = correlator_func(z, zbar_fixed)
        log_G.append(np.log(G))
        log_z.append(np.log(z))

    # Fit log(G) ~ (tau_min / 2) * log(z) + const
    slope, intercept = np.polyfit(log_z, log_G, deg=1)
    tau_min = 2 * slope

    return tau_min


def impose_twist_gap(Delta_grid, J_grid, tau_gap):
    """
    Filter spectrum to impose twist gap.

    Keep only:
    - Conserved currents (   = J + d - 2)
    - Operators with    =    - J     tau_gap
    """
    d = 3  # Dimension

    allowed_ops = []

    for Delta in Delta_grid:
        for J in J_grid:
            tau = Delta - J

            # Conserved current?
            is_conserved = (abs(Delta - (J + d - 2)) < 1e-6)

            if is_conserved or tau >= tau_gap:
                allowed_ops.append((Delta, J))

    return allowed_ops


# Test: Extract identity twist
def test_lightcone_identity():
    """Identity has twist tau = 0."""
    def identity_correlator(z, zbar):
        return 1.0  # Identity contributes constant

    tau_identity = lightcone_expansion(identity_correlator)

    assert abs(tau_identity) < 1e-6, f"Identity twist = {tau_identity},
       expected 0"
    print("    Identity twist = 0 extracted correctly")
```

—

0.3.4 Phase 4: Chaos Bound and Regge Constraints (Months 5-6)

Goal: Implement averaged null energy constraint (ANEC) encoding chaos bound.

```
def averaged_null_energy_operator(Delta_grid, J_grid, crossing_matrix):
    """
    ANEC operator projects onto null energy conditions.

    Chaos bound     ANEC     0 (positive energy along null geodesics).

    Implement as linear constraint in bootstrap.
    """
    # ANEC is a specific linear combination of crossing vectors
    # Reference: Hartman et al. (2016)

    # Simplified: impose positivity of lowest-twist scalar
    anec_weights = np.zeros(len(Delta_grid) * len(J_grid))

    # Weight lowest-twist scalar operators more heavily
    for idx, (Delta, J) in enumerate(zip(Delta_grid, J_grid)):
        if J == 0:  # Scalar
            tau = Delta
            anec_weights[idx] = np.exp(-tau)  # Exponential weight
               favoring low twist

    anec_constraint = anec_weights @ crossing_matrix.T

    return anec_constraint


def impose_chaos_bound(optimization_problem, anec_constraint):
    """
    Add ANEC    0 constraint to SDP.
    """
    # ANEC =     _i  V_i where    comes from functional
    # Require ANEC    0

    optimization_problem.add_constraint(anec_constraint >= 0)

    print("    Chaos bound (ANEC    0) imposed")
```

—

0.3.5 Phase 5: Large-c Bootstrap via SDP (Months 6-8)

Goal: Formulate bootstrap as semidefinite program (SDP); find extremal functional.

```
import cvxpy as cp

def large_c_bootstrap_sdp(Delta_gap, c_value, d=3):
    """
    Bootstrap search for extremal functional.

    Goal: Find   (  , J) such that:
    -       0 for all         _gap  (or allowed by gap assumption)
    -       V < 0 (crossing equation violated)

    If feasible      _gap  is IMPOSSIBLE (bound)
    If infeasible      _gap  is allowed
    """
    # Grid
    Delta_grid = np.linspace(0, 10, 100)
    J_grid = [0, 2, 4, 6, 8]

    # Crossing matrix
    z_points = np.linspace(0.1, 0.9, 20)
    zbar_points = np.linspace(0.1, 0.9, 20)

    V = setup_crossing_matrix(Delta_grid, J_grid, z_points, zbar_points,
        d)

    # Variables: functional
    num_ops = len(Delta_grid) * len(J_grid)
    alpha = cp.Variable(num_ops)
    constraints = []

    # 1. Positivity for          _gap  (excluding conserved)
    for idx, Delta in enumerate(Delta_grid):
        for j_idx, J in enumerate(J_grid):
            op_idx = idx * len(J_grid) + j_idx

            tau = Delta - J
            is_conserved = (abs(Delta - (J + d - 2)) < 1e-3)

            if not is_conserved and tau >= Delta_gap:
                constraints.append(alpha[op_idx] >= 0)

    # 2. Crossing equation:       V < 0 at one point (normalization)
    crossing_eval = alpha @ V

    # Require negative at z=0.5 point (index 10)
    constraints.append(crossing_eval[10] == -1)  # Normalization

    # Require positive derivatives (optional regularization)
    for idx in range(len(z_points) - 1):
        constraints.append(crossing_eval[idx+1] >= crossing_eval[idx] -
           0.1)

    # 3. ANEC constraint (chaos bound)
    anec = averaged_null_energy_operator(Delta_grid, J_grid, V)
    constraints.append(alpha @ anec >= 0)

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

    problem.solve(solver=cp.SCS, eps=1e-8, verbose=True)

    if problem.status == cp.OPTIMAL:
        return {
            'status': 'EXCLUDED',
            'functional': alpha.value,
            'crossing_violation': crossing_eval.value
        }
    else:
        return {
            'status': 'ALLOWED',
            'Delta_gap': Delta_gap
        }


def binary_search_gap_bound(c_value, d=3):
    """
    Binary search for maximum allowed gap.
    """
    Delta_min, Delta_max = 0.0, 5.0
    tolerance = 0.01

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

        print(f"\nTesting  _gap  = {Delta_mid:.3f}...")
        result = large_c_bootstrap_sdp(Delta_mid, c_value, d)

        if result['status'] == 'ALLOWED':
            Delta_min = Delta_mid  # Gap is allowed, try larger
        else:
            Delta_max = Delta_mid  # Gap excluded, try smaller

    print(f"\n=== Maximum allowed gap:  _gap      {Delta_max:.3f} ===")
    return Delta_max


# Run bootstrap
if __name__ == "__main__":
    c = 1000  # Large central charge
    d = 3

    max_gap = binary_search_gap_bound(c, d)

    print(f"Universal bound:  _gap      {max_gap:.3f} for c = {c}, d = {
       d}")
```

Expected Output: For \$d=3\$, literature gives \$
Delta$_{textgap}$
$lesssim(d-2) = 1\$atlargec.$

—

0.3.6 Phase 6: Certificate Generation and Export (Months 8-10)

Goal: Export extremal functional as machine-checkable certificate.

```
import json

def export_bootstrap_certificate(alpha_functional, Delta_grid, J_grid,
                                  crossing_matrix, output_file='
                                     bootstrap_cert.json'):
    """
    Export extremal functional as JSON certificate.
    """
    # Convert to exact rationals where possible
    alpha_rational = [float(x) for x in alpha_functional]  # Simplified

    # Crossing evaluation
    crossing_eval = alpha_functional @ crossing_matrix

    certificate = {
        'metadata': {
            'dimension': 3,
            'central_charge': 1000,
            'bound': ' _gap  upper bound',
            'certificate_version': '1.0'
        },
        'grid': {
            'Delta_grid': Delta_grid.tolist(),
            'J_grid': J_grid
        },
        'functional': {
            'alpha_values': alpha_rational,
            'positivity_check': all(x >= -1e-8 for x in alpha_rational)
        },
        'crossing_violation': {
            'values': crossing_eval.tolist(),
            'max_violation': float(np.max(crossing_eval)),
            'negative_point_exists': any(x < -1e-6 for x in
                crossing_eval)
        },
        'verification': {
            'functional_positive': all(x >= -1e-8 for x in
                alpha_rational),
            'crossing_negative': any(x < -1e-6 for x in crossing_eval)
        }
    }

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

    print(f"    Certificate exported to {output_file}")


def verify_bootstrap_certificate(cert_file):
    """
    Independent verification of bootstrap certificate.
    """
    with open(cert_file, 'r') as f:
        cert = json.load(f)

    print("=== Bootstrap Certificate Verification ===\n")

    # 1. Check functional positivity
    alpha = np.array(cert['functional']['alpha_values'])
    assert cert['functional']['positivity_check'], "Functional not
       positive!"
    print("    Functional is positive on allowed region")

    # 2. Check crossing violation
    crossing = np.array(cert['crossing_violation']['values'])
    assert cert['crossing_violation']['negative_point_exists'], "No
       crossing violation!"
    print(f"    Crossing violated (max = {cert['crossing_violation']['
       max_violation']:.2e})")

    # 3. Recompute crossing matrix (independent check)
    Delta_grid = np.array(cert['grid']['Delta_grid'])
    J_grid = cert['grid']['J_grid']

    z_test = [0.3, 0.5, 0.7]
    zbar_test = [0.4, 0.6, 0.8]

    V_test = setup_crossing_matrix(Delta_grid, J_grid, z_test, zbar_test
       )
    crossing_recomputed = alpha @ V_test

    assert any(x < -1e-6 for x in crossing_recomputed), "Crossing
       verification failed!"
    print("    Independent crossing check passed")

    print("\n=== ALL VERIFICATIONS PASSED ===")
    return True
```

—

0.4 4. Example Starting Prompt

```
You are a theoretical physicist implementing the conformal bootstrap for
    large-c holographic CFTs.
Your task is to derive sharp universal bounds on the twist gap using
   ONLY crossing symmetry,
unitarity, and causality     NO input from bulk gravity or string theory
   .

OBJECTIVE: Find the maximum twist gap   _gap  for stress tensor 4-point
   function in d=3 CFT
at large central charge c >> 1, proving Einstein gravity is constrained
   by consistency alone.

PHASE 1 (Months 1-2): Conformal Block Computation
- Implement conformal blocks g_{  ,J}(z, zbar) for stress tensor
   exchange using:
  * Casimir differential equation (Dolan-Osborn 2011 recursion)
  * Hypergeometric functions for scalar exchange (J=0)
  * Recursion relations for spinning blocks (J > 0)
- Test cases:
  * Identity block (  =0, J=0): verify g_{0,0} = 1
  * Stress tensor block (  =d, J=2): compute at z=0.3, zbar=0.7
- Grid:         [0, 10] with spacing 0.1, J     {0, 2, 4, 6, 8}
- Precision: 100-digit arithmetic via ‘mpmath‘ to avoid accumulation
   errors

PHASE 2 (Months 2-4): Crossing Symmetry
- Formulate crossing equation:
       _  {TT  ,J} [g^{(s)}(z,zbar) - F_{d,J} g^{(t)}(v,u)] = 0
- Build crossing kernel matrix V_{  ,J}(z_i, zbar_i) for 20 20  grid of
   (z, zbar) points
- Verify crossing for known CFTs:
  * Generalized free field (GFF): double-trace spectrum with      ~ 1/c
  * 3D Ising CFT (c     0.95): check against numerical bootstrap data
- Expected: Crossing residual < 10^{-8} for valid theories

PHASE 3 (Months 4-5): Lightcone OPE and Twist Gap
- Extract leading twist in z     0 limit:
  G(z, zbar) ~ z^{ _min /2} F(zbar)
- Identify conserved currents:
  * Identity (   = 0)
  * Stress tensor (   = d-2 = 1 for d=3)
  * Higher-spin currents (   = d-2 if present)
- Impose gap assumption: no non-conserved operators with    <  _gap
- Filter spectrum: keep only          _gap  + J (twist      _gap )

PHASE 4 (Months 5-6): Chaos Bound (Causality)
- Implement averaged null energy condition (ANEC):
      d   T_{    }     0 (averaged energy along null geodesics)
- This encodes chaos bound  _L      2  /   via Regge theory
- Add as SDP constraint:     _i  ANEC_i     0 where    is functional
- Reference: Hartman et al. (2016) "Causality Constraints in Conformal
   Field Theory"
```

```
PHASE 5 (Months 6-8): Bootstrap SDP
- Formulate as semidefinite program (SDP):
  Variables: functional   (  , J) on grid
  Constraints:
    1.        0 for          _gap  (excluding conserved currents)
    2.       V < 0 at one normalization point (crossing violation)
    3. ANEC constraint (chaos bound)
- Solve with CVXPY using SCS or MOSEK solver (tolerance 10^{-8})
- Binary search on  _gap : if SDP feasible     gap excluded; if
   infeasible    gap allowed
- Expected result:  _gap      1.0 for d=3, c         (matches literature
   )

PHASE 6 (Months 8-10): Certificate and Extremal Functional
- Extract extremal functional   *(  , J) from SDP dual
- Verify:
  * Positivity:   *(  , J)      -10^{-8} for all allowed operators
  * Crossing violation:      * V < -10^{-6} at normalization point
- Export JSON certificate:
  * Grid { _i , J_k}
  * Functional values {  *_ik}
  * Crossing kernel evaluated at 10 test points
- Independent verification script: recompute crossing violation from
   certificate

SUCCESS CRITERIA:
- **MVR (Months 3-4)**: Conformal blocks computed, crossing verified for
    GFF, lightcone
  limit extracted (identity twist = 0 confirmed)
- **Strong (Months 7-8)**: Universal bound  _gap      1.0    0.05
   reproduced from literature,
  extremal functional exported, ANEC constraint imposed
- **Publication (Months 9-10)**: New bound with higher-spin currents
   included, comparison
  to known AdS/CFT examples (N=4 SYM), certificate verified
     independently

VERIFICATION PROTOCOL:
1. Conformal blocks: compare to Kos et al. (2014) Table 1 for d=3 scalar
    exchange
2. Crossing: GFF with c=100 should satisfy crossing to 10^{-8}
3. Chaos bound: verify ANEC     0 for all positive-energy states
4. Literature comparison:  _gap  bound for d=3 matches Heemskerk et al.
   (2009)
5. AdS/CFT check: known holographic CFTs (e.g., N=4 SYM at large N)
   respect the bound

EXPORT:
- `bootstrap_certificate.json`: Extremal functional and bound proof
- `conformal_blocks.h5`: Precomputed blocks for        [0,10], J
   {0,2,4,6,8}
- `bootstrap_module.py`: Reusable code for crossing equations, SDP setup

This is a PURE THOUGHT challenge: use ONLY CFT consistency constraints.
NO input from bulk gravity, string theory, or supersymmetry until final
   validation.
```

—

0.5 5. Success Criteria

0.5.1 Minimum Viable Result (MVR) — Months 3-4

Deliverable: Working conformal block code and crossing verification.

Specific Metrics: 1. **Conformal Blocks**: - Scalar blocks (J=0) match literature (Kos et al. 2014) to $10^{-10} - Spinningblocks(J = 2, 4, 6, 8) computedviarecursion, validatedontestcases$

2. **Crossing Symmetry**: - GFF crossing verified: residual ¡ $10^{-8} forc = 100-3DIsingcrossingcheckedag$

3. **Lightcone Extraction**: - Identity twist = 0 extracted correctly (error ¡ 10^{-6}) $- Stresstensortwist = d - 2confirmed$

Certificate: Conformal blocks exported to HDF5, crossing matrix stored as numpy array.

—

0.5.2 Strong Result — Months 7-8

Deliverable: Universal twist gap bound with extremal functional.

Specific Metrics: 1. **Bootstrap Bound**: - _gap 1.0 ± 0.05 for d=3, c $\rightarrow$ (matches literature) - Extremal functional *(, J) extracted from SDP

2. **Chaos Bound**: - ANEC constraint imposed and verified - Comparison to Regge theory predictions

3. **Code Quality**: - SDP solves in ¡ 10 minutes on laptop (MOSEK or SCS) - Grid spacing = 0.05 (100× denser than MVR)

4. **Validation**: - Known holographic CFTs (N=4 SYM, ABJM) respect the bound

Certificate: JSON with extremal functional, independent verification script passes all checks.

—

0.5.3 Publication-Quality Result — Months 9-10

Deliverable: Novel bounds with higher-spin currents, comparison to AdS/CFT.

Specific Metrics: 1. **Novel Contribution**: - Bound on OPE coefficients of spin-4,6,8 currents (if present) - Exclusion region in (c, _gap, _J) parameter space - Proof that certain higher-spin theories are ruled out

2. **Comprehensive Analysis**: - d=2,3,4 dimensions analyzed - Large-c expansion: bounds on 1/c corrections to gravitational couplings - Modular invariance constraints included (thermal correlators)

3. **Validation**: - Comparison to all known holographic CFTs (10+ examples) - Numerical precision: SDP dual gap ¡ $10^{-10} - Formalproof : extremalfunctionalsatisfiesallconstraints$

4. **Publication Targets**: - *Journal of High Energy Physics* (conformal bootstrap) - *Physical Review D* (holography and quantum gravity) - *SciPost Physics* (open access for reproducibility)

Certificate: Complete database of bounds for d=2,3,4 with extremal functionals; Lean/Isabelle formalization of key theorems (stretch goal).

—

0.6 6. Verification Protocol

0.6.1 Automated Checks (Run After Each Phase)

```
def verify_bootstrap_implementation():
    """
    Comprehensive verification suite.
    """
    print("=== Bootstrap Verification Suite ===\n")

    # 1. Conformal blocks
    print("1. Testing Conformal Blocks")
    test_identity_block()
    test_stress_tensor_block()
    test_block_symmetry()

    # 2. Crossing symmetry
    print("\n2. Testing Crossing Symmetry")
    test_gff_crossing()
    test_ising_crossing()

    # 3. Lightcone limits
    print("\n3. Testing Lightcone Limits")
    test_lightcone_identity()
    test_lightcone_stress_tensor()

    # 4. Chaos bound
    print("\n4. Testing Chaos Bound")
    test_anec_positivity()

    # 5. SDP convergence
    print("\n5. Testing SDP Solver")
    test_sdp_toy_problem()

    print("\n=== ALL TESTS PASSED ===")


def test_block_symmetry():
    """Conformal blocks are symmetric under z     zbar for real cross-
       ratios."""
    z, zbar = 0.4, 0.4  # Real point
    Delta, J = 1.5, 2

    block1 = stress_tensor_4pt_block(Delta, J, z, zbar)
    block2 = stress_tensor_4pt_block(Delta, J, zbar, z)

    assert abs(block1 - block2) < 1e-10, "Block not symmetric!"
    print("      Block symmetry (z     zbar) verified")


def test_ising_crossing():
    """3D Ising CFT satisfies crossing (numerical bootstrap data)."""
    # Use known spectrum from bootstrap:    (       0.518),    (
       1.412)
    # OPE coefficients from literature

    Delta_sigma = 0.51815
    Delta_epsilon = 1.41267

    # Simplified: check that crossing is approximately satisfied
    # Full test requires full spectrum

    print("      3D Ising crossing check (simplified)")
```

```
def test_lightcone_stress_tensor():
    """Stress tensor has twist    = d-2."""
    d = 3
    Delta_T, J_T = d, 2

    tau_T = Delta_T - J_T
    tau_expected = d - 2

    assert abs(tau_T - tau_expected) < 1e-10, f"Stress tensor twist = {
        tau_T}, expected {tau_expected}"
    print(f"      Stress tensor twist    = {tau_expected} verified")


def test_anec_positivity():
    """ANEC operator should be positive for all physical states."""
    # Simplified: check that ANEC weights are reasonable
    Delta_grid = np.linspace(1, 5, 10)
    J_grid = [0, 2, 4]

    # Mock crossing matrix
    V = np.random.rand(10, len(Delta_grid) * len(J_grid))

    anec = averaged_null_energy_operator(Delta_grid, J_grid, V)

    # ANEC should be real
    assert np.all(np.isreal(anec)), "ANEC not real!"
    print("      ANEC operator constructed")


def test_sdp_toy_problem():
    """Test SDP solver on simple feasibility problem."""
    # Minimize 0 subject to x     1
    x = cp.Variable()
    constraints = [x >= 1]
    problem = cp.Problem(cp.Minimize(0), constraints)
    problem.solve()

    assert problem.status == cp.OPTIMAL, "SDP solver failed on toy
        problem!"
    assert x.value >= 0.99, "SDP solution incorrect!"
    print("      SDP solver working correctly")
```

Manual Checks: 1. Plot extremal functional (, J) — should be smooth, positive for _gap 2. Compare to literature bounds (Heemskerk 2009, Afkhami-Jeddi 2019) 3. Verify with independent code (e.g., SIMPLEBOOT, JuliBootS)

—

0.7 7. Resources and Milestones

0.7.1 Essential References

Conformal Bootstrap Foundations: - Ferrara, S., Gliozzi, F., & Scherk, J. (1973). "Supergauge multiplets and superfields." *Physical Review D* 8(10): 3303. - Rattazzi, R., et al. (2008). "Bounding scalar operator dimensions in 4D CFT." *Journal of High Energy Physics* 2008(12): 031.

Holographic Bootstrap: - Heemskerk, I., et al. (2009). "Holography from conformal field theory." *Journal of High Energy Physics* 2009(10): 079. - Afkhami-Jeddi, N., et al.

(2019). "Shockwaves from the operator product expansion." *Journal of High Energy Physics* 2019(3): 201.

Chaos and Causality: - Maldacena, J., Shenker, S. H., & Stanford, D. (2016). "A bound on chaos." *Journal of High Energy Physics* 2016(8): 106. - Hartman, T., et al. (2016). "Causality constraints in conformal field theory." *Journal of High Energy Physics* 2016(5): 099.

Numerical Methods: - Simmons-Duffin, D. (2015). "A semidefinite program solver for the conformal bootstrap." *Journal of High Energy Physics* 2015(6): 174. - Kos, F., Poland, D., & Simmons-Duffin, D. (2014). "Bootstrapping the O(N) vector models." *Journal of High Energy Physics* 2014(6): 091.

Software: - SDPB (Simmons-Duffin): high-precision SDP solver for bootstrap - JuliBootS (Julia implementation), PyCFTBoot (Python), Scalar Blocks (Mathematica)

0.7.2 Milestone Checklist

Month 1-2: - [ ] Conformal blocks implemented for d=3, verified against literature - [ ] Hypergeometric function evaluation to 100-digit precision - [ ] Recursion relations for J=2,4,6,8 working

Month 3-4: - [ ] Crossing matrix built for 20×20 (z, zbar) grid - [ ] GFF crossing verified to $10^{-8} - []3DIsingdataloadedandcompared$

Month 5-6: - [ ] Lightcone limit extraction automated - [ ] Twist gap filtering implemented - [ ] ANEC constraint added to SDP

Month 7-8: - [ ] First SDP solved: _gap bound reproduced from literature - [ ] Extremal functional extracted - [ ] Binary search converges to precision 0.01

Month 9-10: - [ ] Higher-spin currents included in analysis - [ ] Bounds for d=2,3,4 computed - [ ] Certificate exported and verified independently - [ ] Comparison to AdS/CFT examples (N=4 SYM, ABJM) - [ ] Draft paper prepared

0.7.3 Common Pitfalls

1. **Numerical Precision**: Conformal blocks have poles/branch cuts; use high-precision arithmetic (100+ digits) to avoid catastrophic cancellation

2. **SDP Solver Tolerance**: Default tolerances $(10^{-4}) are insufficient; required ual gap < 10^{-8} for reliable bounds$

3. **Grid Spacing**: Too coarse → spurious bounds; too fine → SDP intractable. Start with = 0.1, refine near bound

4. **Chaos Bound Implementation**: ANEC is subtle; verify against known CFTs before using in bootstrap

5. **Crossing Kernel Symmetry**: Must respect s t channel symmetry; errors here invalidate entire bootstrap

—

End of PRD 04