Challenge 01: AdS Pure Gravity via the Modular Bootstrap

Pure Thought AI Challenge 01

Pure Thought AI Challenges Project

January 18, 2026

Abstract

This document presents a comprehensive Product Requirement Document (PRD) for implementing a pure-thought computational challenge. The problem can be tackled using only symbolic mathematics, exact arithmetic, and fresh code—no experimental data or materials databases required until final verification. All results must be accompanied by machine-checkable certificates.

Contents

Domain: Quantum Gravity Particle Physics
Difficulty: High
Timeline: 6-12 months
Prerequisites: Conformal field theory, modular forms, semidefinite programming

---

0.1 Problem Statement

0.1.1 Scientific Context

The **AdS/CFT correspondence** (Anti-de Sitter / Conformal Field Theory), discovered by Maldacena in 1997, stands as one of the most profound insights in theoretical physics. It posits an exact duality between quantum gravity in (d+1)-dimensional Anti-de Sitter space and a d-dimensional conformal field theory living on the boundary. For **AdS/CFT**—the three-dimensional bulk with a two-dimensional boundary—this correspondence is particularly tractable due to the infinite-dimensional Virasoro symmetry of 2D CFTs.

Pure gravity in AdS contains only the graviton and no additional matter fields. According to AdS/CFT, such a theory should be dual to an **extremal CFT**: a 2D conformal field theory where the spectrum is maximally sparse, containing only the stress tensor (and its Virasoro descendants) up to a large conformal dimension gap $_gap. For central charge c = 24, this extremal theory is unique and corresponds to the **Monstrous Moonshine** $CFT, whose partition function

The **modular bootstrap** approach exploits the fact that the torus partition function Z() of any 2D CFT must be invariant under the modular group SL(2,), which acts on the modular parameter by Möbius transformations. This **modular invariance**, combined with **unitarity** (all operator degeneracies are non-negative) and **crossing symmetry**, imposes powerful constraints that can be formulated as a linear or semidefinite programming problem. By solving this optimization problem, one can either:

- **Construct** explicit extremal CFTs by finding consistent spectra
- **Rule out** existence via dual certificates proving no solution exists

For c = 24k with k > 1, the question of whether extremal CFTs exist is **open**. If they exist, they would provide consistent quantum gravity theories in AdS at those central charges. If they provably do not exist, this would constrain which gravitational theories are mathematically consistent—shedding light on the **Swampland program** distinguishing low-energy effective theories that can be UV-completed into quantum gravity from those that cannot.

0.1.2 The Core Question

Do extremal 2D CFTs exist for central charge c = 24k (k > 1) with only Virasoro primaries below the gap $_gap/12$?

For k = 1 (c = 24), the Monster CFT provides an explicit example with $_gap = 2. For higher k, no such theories are known, and numerical evidence suggests they may not exist for certain gaps. Pro

0.1.3 Why This Matters

- **Existence: Constructing explicit extremal CFTs would prove pure AdS gravity theories exist at these central charges and potentially reveal new moonshine phenomena connecting number theory and physics**

- **Impossibility: Rigorous no-go theorems would constrain the landscape of quantum gravity theories and support Swampland conjectures about which effective theories can be UV-completed**
- **Method: The modular bootstrap uses only fundamental axioms (modular invariance, unitarity, integrality)—no phenomenological input or experimental data required**
- **Moonshine: Connections between modular forms, sporadic groups, and physics have led to profound mathematical discoveries; extremal CFTs at higher c might reveal new instances**

0.2 Mathematical Formulation

0.2.1 Virasoro Algebra and Characters

A 2D CFT with central charge c is characterized by its Virasoro algebra:

```
[L_m, L_n] = (m - n) L_{m+n} + (c/12) m( m   - 1)  _ {m+n,0}
```

Primary operators |h⟩ satisfy L*0 |h⟩ = h |h⟩ (conformal dimension h) and L*m |h⟩ = 0 for m > 0. The Virasoro character at level h is:

```
 _h (q) = Tr_{V_h} q^{L_0 - c/24}
        = q^{h - c/24} /   _  {n=1}^    (1 - q^n)
        = q^{h - c/24}   (  )^{-1}
```

where q = exp(2i), is the modular parameter, and () is the Dedekind eta function:

```
  (  ) = q^{1/24}   _  {n=1}^    (1 - q^n)
```

The torus partition function is:

```
Z(  ,     ) =  _h  d(h) | _h (  )|
```

where d(h) is the degeneracy of primary operators at conformal dimension h.

0.2.2 Modular Invariance

The partition function must be invariant under the modular group PSL(2,) = SL(2,)/±I. The key generators are:

```
S:         -1/
T:            + 1
```

For holomorphic CFTs (no anti-holomorphic dependence), we have:

```
Z(  ) =  _0 (  ) +  _ {h > 0} d(h)  _h (  )
```

Modular S-transformation relates characters via:

```
 _h (-1/  ) =  _ {h'} S_{h,h'}  _ {h'}(  )
```

For Virasoro characters, the S-matrix is:

```
S_{h,h'} = i exp(-2  i  (hh'))  (approximately, for large c)
```

Modular invariance Z() = Z(-1/) imposes infinitely many linear constraints on the degeneracies d(h).

0.2.3 Extremality Condition

An extremal CFT has no primaries in the gap (0, $_gap$)$except the vacuum$:

```
d(0) = 1  (vacuum)
d(h) = 0  for 0 < h <  _gap
d(h)     0  for h      _gap
```

For c = 24k, a natural gap choice is $_gap = c/12 = 2k, though other gaps can be explored.$

0.2.4 Optimization Problem Formulation

The modular bootstrap can be cast as a linear programming (LP) or semidefinite programming (SDP) feasibility problem:

Primal Problem:

```
Find: {d(h)        , d(h)     0} for h      _gap
Subject to:
  1. Z(  ) - Z(-1/  ) = 0  (modular invariance)
  2. d(h)     0           (unitarity)
  3. d(h)                   (integrality)
```

In practice, we truncate the spectrum at some large h$_max (justified by asymptotic growth bounds) and$

```
Minimize: 0  (feasibility problem)
Variables: d(h) for h     { _gap ,  _gap  + 1, ..., h_max}
Constraints: Modular invariance equations + d(h)     0
```

Dual Problem:

If the primal is infeasible, the LP dual provides a certificate of impossibility: a functional (h) such that:

```
 _h    (h) [modular constraint]_h < 0
  (h)     0 for allowed h
```

This proves mathematically that no solution exists.

0.2.5 Certificate of Correctness

If feasible (extremal CFT exists):

- **Explicit list of degeneracies d(h) for h [*gap, h*max]**
- **Verification: Compute Z() and check |Z() - Z(-1/)| < numerically at many points**
- **Verification: All d(h) are non-negative integers**
- **Export to JSON with exact integer values**

If infeasible (no extremal CFT):

- **SDP/LP dual certificate: functional (h) with exact rational coefficients**
- **Verification: Check certifies infeasibility via Farkas lemma**
- **Export to SMT-LIB or LP certificate format for independent verification**
- **Optionally: Formalize in Lean/Isabelle for machine-checked proof**

0.3 Implementation Approach

0.3.1 Phase 1: Virasoro Characters and Modular Forms (Months 1-2)

Goal: Build high-precision calculator for Virasoro characters and modular transformations.

Key Components:

- **Dedekind eta function:**

```python
from mpmath import mp, exp, pi, I, qfrom, prod

mp.dps = 150  # 150 decimal places

def dedekind_eta(tau: complex) -> complex:
    """
    Compute   (  ) = q^{1/24}   _  {n=1}^    (1 - q^n)

    Uses q-series truncation with error control.
    """
    q = mp.exp(2 * mp.pi * I * tau)

    # Product truncation (error ~ q^{N_max})
    N_max = 100
    product = mp.mpf(1)

    for n in range(1, N_max + 1):
        product *= (1 - q**n)

    eta = q**(mp.mpf(1)/24) * product

    return complex(eta)

def test_eta_modular():
    """
    Verify   (-1/  ) =    (- i  )   (  )
    """
    tau = 0.3 + 0.5j

    eta_tau = dedekind_eta(tau)
    eta_S_tau = dedekind_eta(-1/tau)

    # Expected relation
    expected = mp.sqrt(-I * tau) * eta_tau

    assert abs(eta_S_tau - expected) < 1e-50
    print(f"   modular check passed: error = {abs(eta_S_tau -
        expected)}")
```

- **Virasoro character:**

```python
def virasoro_character(c: float, h: float, tau: complex) ->
    complex:
    """
    Compute  _h (  ) = q^{h - c/24} /   (  )

    Args:
```

```
        c: central charge
        h: conformal dimension
          : modular parameter (Im(  ) > 0)

    Returns:
        Character value  _h (  )
    """
    q = mp.exp(2 * mp.pi * I * tau)

    eta_tau = dedekind_eta(tau)

    chi = q**(h - c/24) / eta_tau

    return complex(chi)

def character_grid(c: float, h_values: List[float], tau: complex)
   -> np.ndarray:
    """
    Compute characters for multiple h values.
    """
    chi = np.array([virasoro_character(c, h, tau) for h in
       h_values], dtype=complex)
    return chi
```

- **Partition function:**

```
from typing import Dict

def partition_function(c: float, spectrum: Dict[float, int], tau:
   complex) -> complex:
    """
    Compute Z(  ) =  _0 (  ) +  _h  d(h)  _h (  )

    Args:
        c: central charge
        spectrum: dictionary {h: d(h)} of degeneracies
        tau: modular parameter

    Returns:
        Partition function Z(  )
    """
    # Vacuum contribution
    Z = virasoro_character(c, 0, tau)

    # Sum over primaries
    for h, dh in spectrum.items():
        if h > 0:
            Z += dh * virasoro_character(c, h, tau)

    return Z

def verify_modular_invariance(c: float, spectrum: Dict[float, int],
                              tau_samples: List[complex], tol:
                                 float = 1e-50) -> bool:
    """
    Verify Z(  ) = Z(-1/  ) at multiple points.
```

```
    """
    for tau in tau_samples:
        Z_tau = partition_function(c, spectrum, tau)
        Z_S_tau = partition_function(c, spectrum, -1/tau)

        error = abs(Z_tau - Z_S_tau)
        if error > tol:
            print(f"Modular invariance violated at   ={tau}:
                error={error}")
            return False

    return True
```

Validation: Test on known modular forms (j-invariant, E, E).

0.3.2 Phase 2: Modular S-Matrix and Constraints (Month 2-3)

Goal: Compute S-matrix $S_{h,h'}$ and formulate modular invariance as linear equations.

S-Matrix Computation:

For large central charge c, the Virasoro S-matrix can be approximated, but for exact results we compute it from:

```
 _h (-1/  ) =  _ {h'} S_{h,h'}  _ {h'}(  )
```

```
def compute_s_matrix(c: float, h_values: List[float], tau_ref: complex
    = 0.1 + 0.5j) -> np.ndarray:
    """
    Compute S-matrix S_{h,h'} by evaluating characters.

    S_{h,h'} = coefficient of  _ {h'}(  ) in expansion of  _h (-1/  )
    """
    N = len(h_values)
    S = np.zeros((N, N), dtype=complex)

    # Compute characters at reference
    chi_tau = np.array([virasoro_character(c, h, tau_ref) for h in
        h_values])

    # Compute characters at -1/
    tau_S = -1 / tau_ref
    chi_S_tau = np.array([virasoro_character(c, h, tau_S) for h in
        h_values])

    # Solve for S: chi_S_tau = S @ chi_tau
    # In practice, use multiple    points and least squares

    for i, h in enumerate(h_values):
        chi_h_S = virasoro_character(c, h, tau_S)

        # Express as linear combination of chi_{h'}(tau_ref)
        # Solve via least squares (overdetermined system with multiple
            )
        # For now, use direct inversion (requires N tau points)
        pass

```

```
    # Simplified: use asymptotic formula for large c
    for i, h in enumerate(h_values):
        for j, hp in enumerate(h_values):
            S[i, j] = I * mp.exp(-2 * mp.pi * I * mp.sqrt(h * hp))

    return S
```

Modular Constraints:
Modular invariance Z() = Z(-1/) gives:

```
 _0 (-1/  ) +  _h  d(h)  _h (-1/  ) =  _0 (  ) +  _h  d(h)  _h (  )
```

Using S-matrix:

```
 _ {h'} S_{0,h'}  _ {h'}(  ) +  _h  d(h)  _ {h'} S_{h,h'}  _ {h'}(  ) =
    _0 (  ) +  _h  d(h)  _h (  )
```

Matching coefficients of $_{h'}()foreachh'$:

```
S_{0,h'} +  _h  d(h) S_{h,h'} =  _ {h',0} + d(h')
```

Rearranging:

```
 _h  [S_{h,h'} -  _ {h,h'}] d(h) =  _ {h',0} - S_{0,h'}
```

This is a system of linear equations in the degeneracies d(h).

```
def setup_modular_constraints(c: float, gap: float, h_max: float,
                              h_values: List[float]) ->
                                 Tuple[np.ndarray, np.ndarray]:
    """
    Set up Ax = b for modular invariance.

    Variables: x = [d(h_1), d(h_2), ..., d(h_N)]
    where h_i     [gap, h_max]

    Constraints: one equation per h' in h_values
    """
    N = len(h_values)

    # Compute S-matrix
    S = compute_s_matrix(c, h_values)

    # Build constraint matrix A and RHS b
    A = np.zeros((N, N), dtype=complex)
    b = np.zeros(N, dtype=complex)

    for i, hp in enumerate(h_values):
        # Equation for h'=hp
        for j, h in enumerate(h_values):
            A[i, j] = S[j, i] - (1 if h == hp else 0)

        # RHS
        b[i] = (1 if hp == 0 else 0) - S[0, i]

    return A, b
```

0.3.3 Phase 3: Linear Programming and Optimization (Months 3-4)

Goal: Solve for non-negative integer degeneracies or certify infeasibility.

Relaxed LP (continuous variables):

```
import cvxpy as cp

def solve_modular_bootstrap_lp(c: float, gap: float, h_max: float) ->
    Dict:
    """
    Solve modular bootstrap as linear program.

    Minimize: 0  (feasibility problem)
    Subject to: A @ d = b, d     0
    """
    h_values = np.arange(gap, h_max + 1, 1.0)
    N = len(h_values)

    A, b = setup_modular_constraints(c, gap, h_max, h_values)

    # Convert to real system (separate real/imaginary parts)
    A_real = np.vstack([A.real, A.imag])
    b_real = np.hstack([b.real, b.imag])

    # Define variables
    d = cp.Variable(N, nonneg=True)

    # Constraints
    constraints = [A_real @ d == b_real]

    # Solve
    problem = cp.Problem(cp.Minimize(0), constraints)
    problem.solve(solver=cp.SCS, verbose=True)

    if problem.status == cp.OPTIMAL:
        spectrum = {h: d.value[i] for i, h in enumerate(h_values)}
        return {
            'status': 'feasible',
            'spectrum': spectrum,
            'dual_certificate': None
        }
    elif problem.status == cp.INFEASIBLE:
        # Extract dual certificate
        dual = constraints[0].dual_value
        return {
            'status': 'infeasible',
            'spectrum': None,
            'dual_certificate': dual
        }
    else:
        return {'status': 'unknown'}
```

Integer Programming:

For exact integrality, use branch-and-bound or rounding + verification:

```
from scipy.optimize import milp, LinearConstraint, Bounds

```

```python
def solve_modular_bootstrap_milp(c: float, gap: float, h_max: float) -> Dict:
    """
    Solve with integer constraints using MILP.
    """
    h_values = np.arange(gap, h_max + 1, 1.0)
    N = len(h_values)

    A, b = setup_modular_constraints(c, gap, h_max, h_values)
    A_real = np.vstack([A.real, A.imag])
    b_real = np.hstack([b.real, b.imag])

    # MILP setup
    constraints = LinearConstraint(A_real, b_real, b_real)
    bounds = Bounds(lb=0, ub=1e6)
    integrality = np.ones(N)  # All variables integer

    result = milp(c=np.zeros(N), constraints=constraints, bounds=bounds,
                  integrality=integrality)

    if result.success:
        spectrum = {h: int(round(result.x[i])) for i, h in
            enumerate(h_values)}
        return {'status': 'feasible', 'spectrum': spectrum}
    else:
        return {'status': 'infeasible'}
```

0.3.4 Phase 4: Extremal CFT Search at c=24k (Months 4-6)

Goal: Systematically search for extremal CFTs at c = 48, 72, 96, ...

Monster CFT Validation (c=24, k=1):

```python
def test_monster_cft():
    """
    Verify we recover the Monster CFT at c=24.

    Known spectrum:
    d(1) = 0 (gap at   =2)
    d(2) = 196884
    d(3) = 21493760
    ...
    """
    c = 24
    gap = 2
    h_max = 10

    result = solve_modular_bootstrap_milp(c, gap, h_max)

    assert result['status'] == 'feasible'

    # Check first few degeneracies
    monster_spectrum = {
        2: 196884,
        3: 21493760,
        4: 864299970
```

```
    }

    for h, d_expected in monster_spectrum.items():
        d_computed = result['spectrum'][h]
        assert abs(d_computed - d_expected) < 1
        print(f"d({h}) = {d_computed} (expected {d_expected})")

    print("Monster CFT validation: PASSED")
```

k=2 Search (c=48):

```
def search_extremal_c48():
    """
    Search for extremal CFT at c=48 with gap   =4.
    """
    c = 48
    gap = 4
    h_max = 20

    print(f"Searching for extremal CFT at c={c}, gap={gap}")

    result = solve_modular_bootstrap_milp(c, gap, h_max)

    if result['status'] == 'feasible':
        print("FOUND FEASIBLE SPECTRUM:")
        for h, d in sorted(result['spectrum'].items()):
            if d > 0.01:  # Only print non-zero
                print(f"  d({h}) = {d}")

        # Verify modular invariance
        tau_samples = [0.1 + 0.5j, 0.2 + 0.8j, 0.5 + 1.0j]
        verified = verify_modular_invariance(c, result['spectrum'],
           tau_samples)
        print(f"Modular invariance verification: {verified}")

        # Export
        export_spectrum(c, gap, result['spectrum'])

    else:
        print("INFEASIBLE: No extremal CFT exists at this gap")

        # Export dual certificate
        if result['dual_certificate'] is not None:
            export_impossibility_certificate(c, gap,
               result['dual_certificate'])

    return result
```

Phase Diagram:

```
def compute_phase_diagram(k_max: int = 5):
    """
    Map out (c, gap) phase diagram: feasible vs infeasible.
    """
    results = {}

    for k in range(1, k_max + 1):
```

```
        c = 24 * k

        for gap in [c/12 - 1, c/12, c/12 + 1]:
            print(f"\nTesting c={c}, gap={gap}")

            result = solve_modular_bootstrap_milp(c, gap,
                h_max=int(c/2))
            results[(c, gap)] = result['status']

            print(f"Result: {result['status']}")

    # Visualize
    import matplotlib.pyplot as plt

    fig, ax = plt.subplots()

    for (c, gap), status in results.items():
        color = 'green' if status == 'feasible' else 'red'
        ax.scatter(c, gap, c=color, s=100)

    ax.set_xlabel('Central charge c')
    ax.set_ylabel('Gap  _gap ')
    ax.set_title('Extremal CFT Phase Diagram')
    plt.savefig('phase_diagram.png')

    return results
```

0.3.5 Phase 5: Dual Certificates and Impossibility Proofs (Months 6-8)

Goal: Extract machine-verifiable certificates when no solution exists.

Certificate Extraction:

```
def extract_dual_certificate(c: float, gap: float, h_max: float) ->
    Optional[np.ndarray]:
    """
    Solve dual LP to get impossibility certificate.

    Dual problem:
    Maximize: b^T y
    Subject to: A^T y     0

    If dual is unbounded, primal is infeasible.
    """
    h_values = np.arange(gap, h_max + 1, 1.0)
    A, b = setup_modular_constraints(c, gap, h_max, h_values)

    A_real = np.vstack([A.real, A.imag])
    b_real = np.hstack([b.real, b.imag])

    M = A_real.shape[0]
    y = cp.Variable(M)

    objective = cp.Maximize(b_real @ y)
    constraints = [A_real.T @ y <= 0]

```

```
    problem = cp.Problem(objective, constraints)
    problem.solve()

    if problem.status == cp.OPTIMAL and problem.value > 1e-6:
        # Found certificate
        return y.value
    else:
        return None

def verify_dual_certificate(A: np.ndarray, b: np.ndarray, y:
    np.ndarray) -> bool:
    """
    Verify that y is a valid dual certificate.

    Check:
    1. A^T y     0
    2. b^T y > 0
    """
    # Check dual feasibility
    dual_slack = A.T @ y
    if not np.all(dual_slack <= 1e-10):
        print("Dual certificate violates feasibility")
        return False

    # Check proves infeasibility
    objective = b.T @ y
    if objective <= -1e-10:
        print(f"Dual objective = {objective}     0, proves
            infeasibility")
        return True
    else:
        print(f"Dual objective = {objective} not sufficient")
        return False
```

Export to SMT-LIB:

```
def export_certificate_smtlib(c: float, gap: float, y: np.ndarray,
    filename: str):
    """
    Export dual certificate to SMT-LIB format for independent
        verification.
    """
    with open(filename, 'w') as f:
        f.write("; Impossibility certificate for extremal CFT\n")
        f.write(f"; c = {c}, gap = {gap}\n")
        f.write("(set-logic QF_LRA)\n\n")

        # Declare dual variables
        for i in range(len(y)):
            f.write(f"(declare-const y{i} Real)\n")

        # Encode A^T y <= 0
        # (implementation details)

        # Encode b^T y > 0
        # (proves infeasibility)

```

```
        f.write("(check-sat)\n")
        f.write("(get-model)\n")

    print(f"Certificate exported to {filename}")
```

0.3.6 Phase 6: Formal Verification and Publication (Months 8-12)

Goal: Formalize results in Lean for machine-checked proofs.

Lean Formalization Template:

```
import Mathlib.Analysis.Complex.Basic
import Mathlib.LinearAlgebra.Matrix.Spectrum

-- Define Virasoro character
def virasoro_character (c h :    ) (   :    ) :     := sorry

-- Modular invariance axiom
axiom modular_invariance (c :    ) (Z :           ) :
  (      , Z    = Z (-1/  ))     ModularInvariant Z

-- Extremal CFT theorem
theorem no_extremal_cft_c48_gap4 :
      (spectrum :           ),
    (    h, 0 < h     h < 4     spectrum h = 0)      -- gap condition
    (    h, spectrum h     0)                       -- unitarity
      (ModularInvariant (partition_function 48 spectrum)) := by
  intro spectrum h_gap h_unit
  -- Proof using dual certificate
  sorry
```

Certificate Verification Code:

```
def generate_lean_proof(c: float, gap: float, certificate: np.ndarray,
   filename: str):
    """
    Generate Lean proof script from dual certificate.
    """
    with open(filename, 'w') as f:
        f.write("-- Impossibility proof for extremal CFT\n")
        f.write(f"-- c = {c}, gap = {gap}\n\n")

        # Define certificate as explicit vector
        f.write("def dual_certificate : Vector     := ")
        f.write(f"![{', '.join(map(str, certificate))}]\n\n")

        # State and prove theorem
        f.write("theorem extremal_cft_impossible :\n")
        f.write(f"  no_extremal_CFT {c} {gap} := by\n")
        f.write("  apply dual_certificate_implies_infeasible\n")
        f.write("  exact dual_certificate\n")
        f.write("  -- Verification steps\n")
        f.write("  sorry\n")

    print(f"Lean proof template written to {filename}")
```

0.4 Example Starting Prompt

```
I need you to implement a complete modular bootstrap solver for 2D CFTs to search
for extremal theories corresponding to pure  A d S  gravity. This is a research-level
problem in quantum gravity that requires exact arithmetic and rigorous certificates.

SCIENTIFIC GOAL:
Determine whether extremal CFTs exist at central charge c = 48 (and higher c = 24k)
with only the vacuum operator below a conformal dimension gap  _gap  c/12.

For c=24, the Monster CFT is the unique extremal theory. For c=48, existence is unknown.
We will either construct an explicit spectrum or prove impossibility via dual certificates.

IMPLEMENTATION PHASES:

PHASE 1 - Virasoro Characters (Weeks 1-2):
1. Implement dedekind_eta(tau) computing  (  ) = q^{1/24}  (1-q^n) to 100+ digit precision
   Use mpmath with mp.dps = 150

2. Implement virasoro_character(c, h, tau) computing  _h (  ) = q^{h-c/24}/  (  )

3. Test modular transformation: verify  (-1/  ) =  (- i  )  (  ) numerically

4. Implement partition_function(c, spectrum, tau) computing Z(  ) =  _h  d(h)  _h (  )

PHASE 2 - Modular Constraints (Weeks 2-4):
5. Compute modular S-matrix S_{h,h'} relating  _h (-1/  ) =  S_{h,h'}  _ {h'}(  )
   For large c, use asymptotic formula S_{h,h'}  i exp(-2  i  (hh'))

6. Formulate modular invariance Z(  ) = Z(-1/  ) as linear system A @ d = b
   where d = [d( _gap ), d( _gap +1), ..., d(h_max)]

7. Export constraint matrix to exact rational arithmetic

PHASE 3 - Linear Programming (Weeks 4-8):
8. Solve LP relaxation: find d  0 satisfying Ad = b using cvxpy
   Start with c=24, gap=2 to validate against Monster CFT

9. Verify solution: d(2) should equal 196884 for Monster

10. If LP is infeasible, extract dual certificate proving impossibility

PHASE 4 - Integer Programming (Weeks 8-12):
```

```
11. Implement integer rounding: round LP solution and verify constraints

12. If rounding fails, use MILP solver (scipy.optimize.milp) with
    integrality constraints

13. For c=24, confirm exact Monster spectrum:
    d(2) = 196884, d(3) = 21493760, d(4) = 864299970

PHASE 5 - k=2 Search (Weeks 12-16):
14. Apply solver to c=48, gap=4, h_max=20

15. If feasible: export spectrum, verify modular invariance at 10+
    random    points

16. If infeasible: extract and export dual certificate in SMT-LIB format

17. Verify certificate independently using Z3 or external LP solver

PHASE 6 - Classification (Weeks 16-24):
18. Repeat for k=3,4,5 (c=72,96,120)

19. Try multiple gap values for each c

20. Build phase diagram: plot (c, gap) colored by feasible/infeasible

21. Export all certificates to certificates/ directory

TECHNICAL REQUIREMENTS:
- Use mpmath with mp.dps >= 150 for all character computations
- All degeneracies must be exact non-negative integers
- Modular invariance verified to |Z(  ) - Z(-1/  )| < 10^{-50}
- Export certificates in JSON (spectra) and SMT-LIB (impossibility
    proofs)
- Every result must include a machine-checkable certificate

SUCCESS CRITERIA:
MINIMUM (3 months): Monster CFT reproduced, one new result for c=48
STRONG (6 months): Complete results for k=2,3,4 with verified
    certificates
PUBLICATION (12 months): k up to 10, Lean formalization, novel CFTs or
    no-go theorems

START HERE:
Begin by implementing Phase 1. Write dedekind_eta() and test it against
    known values.
Then move to virasoro_character(). Do not proceed to Phase 2 until
    Phase 1 passes
all numerical tests to 100+ digit precision.
```

0.5 Success Criteria

0.5.1 Minimum Viable Result (3-4 months)

Infrastructure validated:

- **Virasoro character calculator accurate to 100+ decimal digits**
- **Modular S-transformation verified numerically with error < 10**
- **Successfully reproduce Monster CFT spectrum at c=24: d(2)=196884, d(3)=21493760, d(4)=864299970**

One new rigorous result:

- **Either: Feasible spectrum for c=48 at gap =4 with all degeneracies verified as non-negative integers**
- **Or: Dual certificate proving impossibility of extremal CFT at c=48 for gap =4**
- **Certificate exported to machine-verifiable format (JSON for spectrum, SMT-LIB for impossibility)**

0.5.2 Strong Result (6-8 months)

Multiple cases resolved:

- **Complete results for k = 2, 3, 4 (c = 48, 72, 96)**
- **For each: either explicit spectrum with d(h) verified or impossibility certificate**
- **All modular invariance constraints satisfied to machine precision (|Z() - Z(-1/)| < 10)**

Rigorous certificates:

- **All spectra exported with exact integer degeneracies in JSON format**
- **All impossibility proofs exported as LP dual certificates in SMT-LIB**
- **Independent verification: certificates checked by Z3 or external LP solver**
- **Documentation of certificate format and verification procedure**

Phase diagram initiated:

- **Scan over gaps [c/12 - 2, c/12 + 2] for each c**
- **Phase diagram plotting (c,) with feasible/infeasible regions identified**
- **Patterns or bounds on maximum gap for extremality observed**

0.5.3 Publication-Quality Result (9-12 months)

Comprehensive classification:

- **Results for k up to 10 (c up to 240)**
- **Refined gap scans: in increments of 0.1 near boundaries**
- **Phase diagram revealing clear structure (if any) in (c,) space**
- **Database of all extremal CFTs found or impossibility certificates generated**

Formal verification:

- **Key impossibility theorems formalized in Lean 4 or Isabelle/HOL**
- **Dual certificates imported and verified in proof assistant**
- **Machine-checked proofs: "No extremal CFT exists at c=48 with gap =4" (or similar)**
- **Formal specification of modular invariance and extremality axioms**

Novel insights:

- **New extremal CFTs discovered beyond Monster (if they exist), or**
- **Systematic impossibility theorems: "For c=24k, k>1, no extremal CFT exists for < c/12" (if true)**
- **Connections to moonshine: check if any found CFTs exhibit sporadic group symmetries**
- **Comparison with holographic bounds: verify extremal CFTs saturate known bounds on gap from AdS gravity**

Publication-ready artifacts:

- **ArXiv preprint with full results and certificate repository**
- **Public database: spectra or impossibility proofs for all (c,) pairs tested**
- **Lean formalization repository with verified theorems**
- **Reproducible Jupyter notebooks for all computations**

0.6 Verification Protocol

0.6.1 Automated Checks

For each claimed result, the following checks must pass:
If claiming feasibility (extremal CFT exists):

```
def verify_extremal_cft(c: float, gap: float, spectrum: Dict[float,
    int],
                        tau_samples: Optional[List[complex]] = None)
                            -> Dict:
    """
    Comprehensive verification of extremal CFT spectrum.

    Returns:
        Dictionary with verification results and error metrics
    """
    if tau_samples is None:
        # Default: 10 random points in fundamental domain
        tau_samples = [0.05 + 0.5j, 0.2 + 0.8j, 0.5 + 1.0j,
```

```
                        -0.4 + 0.7j, 0.3 + 1.2j] * 2

    results = {
        'integrality_passed': True,
        'unitarity_passed': True,
        'gap_passed': True,
        'modular_invariance_passed': True,
        'max_error': 0.0
    }

    # 1. Check integrality
    for h, d in spectrum.items():
        if not isinstance(d, int) or d < 0:
            results['integrality_passed'] = False
            print(f"FAIL: d({h}) = {d} is not a non-negative integer")

    # 2. Check unitarity (redundant with integrality check, but
       explicit)
    if not all(d >= 0 for d in spectrum.values()):
        results['unitarity_passed'] = False

    # 3. Check gap condition
    if any(0 < h < gap for h in spectrum.keys()):
        results['gap_passed'] = False
        print(f"FAIL: Primaries found in gap (0, {gap})")

    # 4. Check modular invariance
    for tau in tau_samples:
        Z_tau = partition_function(c, spectrum, tau)
        Z_S_tau = partition_function(c, spectrum, -1/tau)

        error = abs(Z_tau - Z_S_tau)
        results['max_error'] = max(results['max_error'], error)

        if error > 1e-30:
            results['modular_invariance_passed'] = False
            print(f"FAIL: Modular invariance at   ={tau}:
               error={error}")

    # Overall status
    all_passed = all([
        results['integrality_passed'],
        results['unitarity_passed'],
        results['gap_passed'],
        results['modular_invariance_passed']
    ])

    results['status'] = 'VERIFIED' if all_passed else 'FAILED'

    return results

def export_verified_spectrum(c: float, gap: float, spectrum:
    Dict[float, int],
                             verification: Dict, filename: str):
    """
    Export spectrum with verification metadata to JSON.
```

```
    """
    import json
    from datetime import datetime

    data = {
        'central_charge': c,
        'gap': gap,
        'spectrum': {str(h): int(d) for h, d in spectrum.items()},
        'verification': verification,
        'timestamp': datetime.now().isoformat(),
        'precision': f'{mp.dps} decimal digits'
    }

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

    print(f"Spectrum exported to {filename}")
```

If claiming infeasibility:

```
def verify_impossibility_certificate(c: float, gap: float, h_max: float,
                                     dual_certificate: np.ndarray) ->
                                        bool:
    """
    Verify that dual certificate proves infeasibility.

    Certificate y must satisfy:
    1. A^T y     0  (dual feasibility)
    2. b^T y > 0  (proves primal infeasible via Farkas lemma)

    Returns:
        True if certificate is valid
    """
    h_values = np.arange(gap, h_max + 1, 1.0)
    A, b = setup_modular_constraints(c, gap, h_max, h_values)

    A_real = np.vstack([A.real, A.imag])
    b_real = np.hstack([b.real, b.imag])

    y = dual_certificate

    # Check dual feasibility: A^T y     0
    dual_slack = A_real.T @ y

    if not np.all(dual_slack <= 1e-8):
        max_violation = np.max(dual_slack)
        print(f"FAIL: Dual feasibility violated, max violation =
            {max_violation}")
        return False

    # Check proves infeasibility: b^T y > 0
    # (For standard LP, opposite sign convention may apply)
    objective = b_real @ y

    if objective <= -1e-10:
        print(f"SUCCESS: Dual objective = {objective} < 0 proves
            infeasibility")
```

```
        return True
    else:
        print(f"FAIL: Dual objective = {objective} does not prove
            infeasibility")
        return False

def export_impossibility_certificate(c: float, gap: float, certificate:
    np.ndarray,
                                     filename: str):
    """
    Export dual certificate to JSON and SMT-LIB formats.
    """
    import json

    # JSON format
    json_file = filename + '.json'
    data = {
        'central_charge': c,
        'gap': gap,
        'result': 'infeasible',
        'dual_certificate': certificate.tolist(),
        'verification': 'Use verify_impossibility_certificate()'
    }

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

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

    # SMT-LIB format (for independent verification)
    smtlib_file = filename + '.smt2'
    export_certificate_smtlib(c, gap, certificate, smtlib_file)
```

0.6.2 Human-Reviewable Artifacts

- **Spectrum file (if feasible):** spectrum*cc*gap*gap*.json

```
{
  "central_charge": 48,
  "gap": 4,
  "spectrum": {
    "4": 12345678,
    "5": 98765432,
    "6": 111222333
  },
  "verification": {
    "status": "VERIFIED",
    "max_error": 1.2e-51,
    "integrality_passed": true,
    "modular_invariance_passed": true
  },
  "timestamp": "2026-01-17T10:30:00",
  "precision": "150 decimal digits"
}
```

- **Impossibility certificate (if infeasible):** `certificate`$ccgapgap$`.smt2`
- **SMT-LIB format encoding dual feasibility and infeasibility proof**
- **Can be verified independently using Z3:** `z3 certificate`$c48gap4$`.smt2`
- **Proof script:** `proof`$ccgapgap$`.lean`
- **Formal Lean 4 proof importing certificate and verifying impossibility**
- **Type-checks in Lean to confirm mathematical correctness**
- **Phase diagram:** `phase`$_diagram.png$
- **Plot of (c,) space with color-coded feasible/infeasible regions**
- **Includes all tested points with markers**

0.7 Resources References

0.7.1 Essential Papers

- **Hartman, Keller, Stoica (2014): "Universal Spectrum of 2d Conformal Field Theory in the Large c Limit"**

[arXiv:1405.5137] - Establishes universal bounds on spectrum using modular bootstrap at large c

- **Afkhami-Jeddi, Cohn, Hartman, Tajdini (2020): "Free Partition Functions and an Averaged Holographic Duality"**

[arXiv:2006.04839] - Uses modular bootstrap to constrain averaged CFT ensembles

- **Collier, Lin, Yin (2019): "Modular Bootstrap Revisited"**

[arXiv:1608.06241] - Systematic treatment of modular constraints for rational CFTs

- **Hellerman (2011): "A Universal Inequality for CFT and Quantum Gravity"**

[arXiv:0902.2790] - Lower bound on gap using modular invariance

- **Friedan, Keller, Yin (2013): "A Remark on AdS/CFT for the Extremal Virasoro Partition Function"**

[arXiv:1312.1536] - Analysis of extremal CFTs and connections to AdS gravity

0.7.2 Code Libraries

- **mpmath: Arbitrary precision arithmetic in Python - `pip install mpmath`**
- **SymPy: Symbolic mathematics - `pip install sympy`**
- **CVXPY: Convex optimization with SDP/LP solvers - `pip install cvxpy`**
- **SciPy: Scientific computing including MILP solvers - `pip install scipy`**
- **Lean 4: Proof assistant for formal verification - https://lean-lang.org**

0.7.3 Mathematical Background

- **Modular Forms: Serre's "A Course in Arithmetic", Diamond Shurman "A First Course in Modular Forms"**
- **Virasoro Algebra: Di Francesco et al. "Conformal Field Theory" (Yellow Book)**
- **Optimization: Boyd Vandenberghe "Convex Optimization" (Chapter on LP duality)**
- **AdS/CFT: Aharony et al. "Large N Field Theories, String Theory and Gravity" [arXiv:hep-th/9905111]**

0.7.4 Key Concepts to Master

- **Dedekind eta function and its modular transformation properties**
- **Virasoro minimal models and character formulas**
- **Modular group PSL(2,) and fundamental domain**
- **Linear programming duality and Farkas lemma (for impossibility certificates)**
- **q-series and asymptotic expansions for characters**
- **Partition functions and genus expansion in CFT**

0.8 Common Pitfalls How to Avoid Them

0.8.1 Numerical Precision Issues

Problem: Modular invariance appears satisfied due to rounding errors; false positives in feasibility

Solution:

- **Use `mpmath` with `mp.dps = 150` or higher for all character computations**
- **Verify modular invariance to at least 50 decimal digits: |Z() - Z(-1/)| $<$ 10**
- **Cross-check with multiple points in fundamental domain**

0.8.2 Fake Spectra from LP Relaxation

Problem: LP solution has d(h) = 123.7 (non-integer) accepted as valid
Solution:

- **Always enforce strict integrality using MILP or branch-and-bound**
- **Round and verify: if rounding LP solution, check all constraints still satisfied**
- **Export only exact integer degeneracies**

0.8.3 Incomplete or Invalid Dual Certificates

Problem: Dual certificate extracted but doesn't rigorously prove infeasibility
Solution:

- **Verify certificate satisfies $\mathbf{A}^T y 0 and b^T y > 0 (or appropriate sign convention)$**
- **Use exact rational arithmetic for certificate verification**
- **Export to SMT-LIB and verify independently with Z3**

0.8.4 Truncation Effects in Spectrum

Problem: Setting $\mathbf{h}_max too small misses important high - dimension operators$
Solution:

- **Start with $\mathbf{h}_max = c (usually sufficient for extremal CFTs)$**
- **Check sensitivity: increase $\mathbf{h}_max and verify solution doesn't change$**
- **Use asymptotic bounds on degeneracies to justify truncation**

0.8.5 S-Matrix Approximation Errors

Problem: Using asymptotic S-matrix formula $\mathbf{S}_{h,h'} i exp(-2i(hh'))$ introduces errors
Solution:

- **Compute S-matrix exactly by evaluating characters at multiple points**
- **For large c, asymptotic formula is accurate; verify error $< 10^{1}$**
- **Cross-check S-matrix satisfies unitarity: S S† = I**

0.8.6 Confusing Modular S and T Transformations

Problem: Implementing only S: → -1/ but ignoring T: → +1
Solution:

- **S and T generate full modular group; both must be satisfied**
- **For extremal CFTs, T-invariance is automatic (weights in), but verify explicitly**
- **Test full modular orbit: check Z() = Z() for multiple PSL(2,)**

0.9 Milestone Checklist

Infrastructure (Months 1-2):

Dedekind eta function () implemented with 100+ digit precision

Modular transformation (-1/) = (-i) () verified numerically

Virasoro character $_h()calculatortestedonknownvalues$

Partition function Z() builder with arbitrary spectrum input

Modular S-matrix $\mathbf{S}_{h,h'}computedandverifiedforunitarity$

Validation (Month 2):

Monster CFT (c=24, gap=2) spectrum reproduced exactly:

d(2) = 196884

d(3) = 21493760

d(4) = 864299970

Modular invariance of Monster partition function verified to 10

Optimization Solvers (Months 2-3):

LP relaxation solver (cvxpy) working and tested

MILP solver (scipy.optimize.milp) enforcing integrality

Dual certificate extraction from infeasible LP implemented

Certificate verification functions tested

New Results (Months 3-6):

c=48, gap=4: Result obtained (feasible or impossibility certificate)

c=48 result verified independently

c=72, gap=6: Result obtained

c=96, gap=8: Result obtained

All certificates exported to JSON/SMT-LIB formats

Classification (Months 6-9):

Phase diagram for k=1 to 5 complete

Gap scan: multiple values tested for each c

Database of all spectra and impossibility proofs assembled

Visualization: (c,) phase diagram plotted

Formal Verification (Months 9-12):

Lean 4 formalization of modular invariance begun

First impossibility theorem formalized and type-checked in Lean

All dual certificates imported and verified in proof assistant

Publication draft with machine-checkable proofs prepared

Publication (Month 12):

ArXiv preprint submitted with full results

Public repository with all certificates and verification code

Reproducible Jupyter notebooks for all computations

All certificates publicly available and independently verified

---

Next Steps:

Begin with Phase 1 infrastructure. Implement `dedekindeta()` `and test it against known values (e.g., (i) =` $^{1/4}/(1/4)^{1/2}$).$Thenimplement$`virasoro`$character()andverifynumericalvalues$
$digitprecision.Validatethoroughlyonthe MonsterCFT(k = 1)beforeattemptinganynewcases(k2).$

`Focus on building robust, high-precision code with comprehensive testing. Every result must be accompanied by a machine-verifiable certificate-either an explicit spectrum or a dual certificate proving impossibility.`