PRD 08: Swampland via Modularity Higher-Form Symmetries

Pure Thought AI Challenge 08

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
Timeline: 9-12 months
Difficulty: High
Prerequisites: Modular forms, CFT, algebraic topology, group cohomology

---

0.1 1. Problem Statement

0.1.1 Scientific Context

The **Swampland Program** aims to identify which low-energy effective field theories (EFTs) can arise from consistent theories of quantum gravity. Not all quantum field theories that appear consistent (unitary, local, Lorentz-invariant) can be UV-completed into quantum gravity—those that cannot are said to lie in the "swampland."

Recent work suggests that **modular invariance** and **higher-form global symmetries** provide powerful constraints:

- **Modular Bootstrap**: In 2D CFTs with central charge c, modular invariance of the partition function Z() on the torus severely restricts the spectrum. For rational CFTs, the modular S-matrix must be unitary and symmetric.
- **Higher-Form Symmetries**: Modern perspectives on symmetry include p-form global symmetries that act on extended objects (strings, membranes). In theories with compact dimensions, these symmetries are tied to the cohomology of the compactification manifold.
- **Anomaly Matching**: 't Hooft anomaly matching for higher-form symmetries places constraints on the IR spectrum. In theories coupled to gravity, certain global symmetries must be absent or gauged (completeness hypothesis).
- **Cobordism Obstructions**: Topological field theories (TFTs) classified by cobordism groups can have 't Hooft anomalies that obstruct their coupling to gravity. The triviality of certain bordism groups in gravitational theories provides swampland constraints.

0.1.2 Core Question

Can we use modular invariance, higher-form symmetry constraints, and cobordism arguments to prove that certain CFT partition functions cannot arise from consistent quantum gravity theories?

Specifically:

- Given a putative 2D CFT partition function Z(), can we certify that it violates gravitational consistency?
- Can we enumerate all rational CFTs at low central charge that are compatible with quantum gravity?
- Can we derive swampland bounds on couplings in higher-dimensional EFTs from CFT constraints?

0.1.3 Why This Matters

Theoretical Impact:

- Provides rigorous, computer-verifiable tests of the swampland program
- Connects abstract CFT mathematics to fundamental questions about quantum gravity
- May reveal universal patterns in the landscape of string compactifications

Practical Benefits:

- Establishes automated tools for checking CFT consistency with gravity
- Generates finite databases of "gravity-compatible" CFTs
- Provides constraints for phenomenological model building beyond the Standard Model

Pure Thought Advantages:

- Modular forms have exact symbolic representations
- Character theory and representation theory are purely algebraic
- Cobordism groups can be computed exactly
- No experimental data required—only mathematical consistency

---

0.2 2. Mathematical Formulation

0.2.1 Problem Definition

Given:

- A candidate 2D CFT with central charges (c, c)
- A partition function Z(,) on the torus T^2
- Assumed modular invariance: Z(· , ·) = Z(,) for SL(2,)
- Assumed unitarity: spectrum contains only positive norm states

We seek to determine:
Is this CFT consistent with quantum gravity?
This breaks down into several computable checks:

Check 1: Modular Invariance Certificate The partition function must decompose as:

```
Z(   ,      ) =   _ {i,j} n_{ij}   _i (   )    _j   (     )
```

where:

- $_i()areholomorphicVirasorocharactersatcentralchargec$
- $_j()areantiholomorphiccharactersatcentralchargec$
- n$_{ij}0arenon-negativeintegermultiplicities$

Under modular S-transformation (→ -1/):

```
 _i  (-1/   ) =   _k   S_{ik}   _k  (   )
```

The S-matrix must be:

- **Unitary**: S S† =
- **Symmetric**: S = S
- **Satisfies** (ST)^3 = S^2, where T is the T-matrix (→ +1)

Certificate: Extract the S-matrix from character transformations, verify unitarity and symmetry using exact rational arithmetic.

Check 2: Global Symmetry Constraints Quantum gravity forbids exact global symmetries. Any apparent global symmetry G must be:

- **Gauged** (becoming a dynamical gauge symmetry), or
- **Explicitly broken** by quantum effects

For a CFT with symmetry group G:

- Compute the 't Hooft anomaly H$^{d+1}(BG, U(1))$
- If is non-trivial, G must be gauged or broken

Certificate: Compute group cohomology H$^*(BG, U(1))symbolically.IfpartitionfunctionexhibitsexactG-$ $symmetrywithnon-trivialanomaly, flagasinconsistent.$

Check 3: Cobordism Constraints Certain topological phases are obstructed in quantum gravity. For a d-dimensional CFT:

- Compute the bordism group $^{Spin}_d relevanttotheCFT'ssymmetryclass$
- Check if the CFT defines a non-trivial element of this group
- If so, verify that this element is trivialized when coupled to gravity

Example: For d=2, $^{Spin}2 = 2.Non-trivialelementscorrespondtotheorieswithgravitationalanomaly.The$
Certificate: Compute bordism invariants using spectral sequences or direct calculation. Extract the relevant characteristic classes from CFT data.

Check 4: Spectrum Positivity For gravity-compatible CFTs:

- The lightest non-vacuum primary must satisfy h c/12 (cf. extremality)
- The degeneracy of states at conformal weight h must grow as exp(2(ch/6))
- Twist gaps (J)   *where* is the "graviton twist"

Certificate: Extract primary spectrum (h*i*, *h*i), compute gaps, verify Cardy growth.

0.2.2 Input/Output Specification

Input:

```
class CFTData:
    c: Fraction                # Central charge (rational)
    c_bar: Fraction            # Antiholomorphic central charge
    characters: List[QSeries]  # Holomorphic characters as q-series
    characters_bar: List[QSeries]  # Antiholomorphic characters
    partition_matrix: Matrix[int]  # Multiplicities n_{ij}
    symmetry_group: Optional[FiniteGroup]  # Global symmetry (if any)
```

Output:

```
class SwamplandCertificate:
    is_consistent: bool
    violations: List[str]  # Human-readable violation messages

    # Certificates
    s_matrix: Matrix[AlgebraicNumber]  # Exact S-matrix
    s_unitarity_error: Fraction  # ||S  S   -     ||, should be 0

    anomaly_class: Optional[CohomologyElement]  # 't Hooft anomaly
    bordism_class: Optional[BordismElement]  # Cobordism invariant

    spectrum: List[Tuple[Fraction, Fraction, int]]  # (h, h  ,
        degeneracy)
    min_gap: Fraction  # Minimum spectral gap

    # Proof artifacts
    drat_proof: Optional[Path]  # If SAT-based checks used
    sdp_certificate: Optional[Path]  # If SDP used for bounds
```

0.3 3. Implementation Approach

0.3.1 Phase 1: Modular Forms and Character Library (Months 1-2)

Build infrastructure for exact modular arithmetic:

```
from sympy import Rational, Symbol, exp, I, pi, sqrt
from sympy.abc import q
from typing import List, Tuple
import mpmath as mp

class QSeries:
```

```python
    """Power series in q = exp(2  i  ) with exact coefficients."""

    def __init__(self, coeffs: List[Rational], offset: Rational):
        """
        Represents  _ {n=0}^    coeffs[n] q^(offset + n)
        """
        self.coeffs = coeffs
        self.offset = offset  # Starting power (can be negative for
            poles)

    def evaluate(self, tau: complex, terms: int = 100) -> complex:
        """Evaluate at specific    using high precision."""
        mp.dps = 50
        q_val = mp.exp(2 * mp.pi * 1j * tau)
        result = mp.mpc(0)
        for n, c in enumerate(self.coeffs[:terms]):
            result += float(c) * q_val**(float(self.offset) + n)
        return result

    def modular_transform_S(self) -> 'QSeries':
        """Apply S:         -1/   transformation."""
        # Implement using Poisson resummation for theta functions
        # Or lookup table for Virasoro characters
        raise NotImplementedError

def virasoro_character(c: Rational, h: Rational, max_terms: int = 200)
    -> QSeries:
    """
    Compute Virasoro character  _h (q) at central charge c.

     _h (q) = q^{h - c/24} Tr_{V_h}(q^{L_0 - c/24})
            = q^{h - c/24} /   (q) * (...)

    where   (q) = q^{1/24}   (1 - q^n) is Dedekind eta.
    """
    # For minimal models, use Kac table
    # For general c, use recursion relations for Virasoro descendants

    coeffs = []
    # Compute using Verma module structure
    # Include null vector relations for degenerate representations

    return QSeries(coeffs, h - c/24)

def dedekind_eta_qseries(max_terms: int = 200) -> QSeries:
    """  (q) = q^{1/24}  _ {n=1}^    (1 - q^n)"""
    coeffs = partition_function_coefficients(max_terms)
    return QSeries(coeffs, Rational(1, 24))
```

Validation: Reproduce known S-matrices for minimal models (c = 1 - 6/m(m+1) for m=3,4,5...).

0.3.2 Phase 2: Modular S-Matrix Extraction (Months 2-4)

Implement modular transformations and extract the S-matrix:

```python
from sympy import Matrix, simplify, algebraic_number
from sympy.polys.numberfields import AlgebraicNumber

def extract_s_matrix(characters: List[QSeries],
                     c: Rational) -> Matrix[AlgebraicNumber]:
    """
    Compute modular S-matrix from character transformations.

      _i (-1/  ) =  _j  S_{ij}  _j (  )

    Returns exact algebraic S-matrix.
    """
    n = len(characters)
    S = Matrix(n, n, lambda i, j: AlgebraicNumber(0))

    # Evaluate characters at specific    values
    # Use numerical evaluation + PSLQ to find exact algebraic relations

    tau_samples = [
        mp.mpc(0, 1),  #    = i
        mp.mpc(0.5, sqrt(3)/2),  #    = exp(2 i /3)
        # ... more samples
    ]

    for i, chi_i in enumerate(characters):
        # Compute chi_i(-1/  )
        transformed_values = [chi_i.evaluate(-1/tau) for tau in
            tau_samples]

        # Express as linear combination of chi_j(  )
        char_values = [[chi_j.evaluate(tau) for tau in tau_samples]
                       for chi_j in characters]

        # Solve linear system to find S_{ij}
        # Use PSLQ to recognize algebraic numbers
        for j in range(n):
            S[i, j] = pslq_algebraic(transformed_values, char_values[j])

    return S

def verify_s_matrix_unitarity(S: Matrix) -> Rational:
    """
    Check ||S  S   -     ||_Frobenius using exact arithmetic.

    Returns 0 if S is exactly unitary.
    """
    S_dagger = S.conjugate().transpose()
    product = S * S_dagger
    identity = Matrix.eye(S.rows)

    error_matrix = product - identity
    error = sqrt(sum(abs(simplify(x))**2 for x in error_matrix))

    return Rational(0) if error == 0 else error
```

Validation: Check $(ST)^3 = S^2$ for minimal models, verify unitarity exactly.

0.3.3 Phase 3: Higher-Form Symmetry Analysis (Months 4-6)

Implement tools for detecting and classifying global symmetries:

```python
from sympy.combinatorics import PermutationGroup
from sympy.polys.polytools import groebner

class FiniteGroup:
    """Finite group specified by generators and relations."""
    def __init__(self, generators: List, relations: List):
        self.generators = generators
        self.relations = relations

    def cohomology_class(self, degree: int) -> 'CohomologyElement':
        """Compute H^d(BG, U(1)) using spectral sequences."""
        # Implement Lyndon-Hochschild-Serre spectral sequence
        # Or use explicit cochain complexes for small groups
        raise NotImplementedError

def detect_global_symmetry(partition_matrix: Matrix[int],
                           s_matrix: Matrix) -> Optional[FiniteGroup]:
    """
    Detect global symmetry group from partition function structure.

    Symmetries permute primaries while preserving n_{ij}.
    """
    n = partition_matrix.rows

    # Find permutation group preserving partition_matrix
    # A symmetry    satisfies: n_{  (i),  (j)} = n_{i,j}

    generators = []
    for perm in permutation_candidates(n):
        if preserves_matrix(perm, partition_matrix):
            generators.append(perm)

    if not generators:
        return None

    G = PermutationGroup(generators)
    return FiniteGroup.from_permutation_group(G)

def compute_anomaly(G: FiniteGroup, cft_data: CFTData) ->
    CohomologyElement:
    """
    Compute 't Hooft anomaly         H  (BG, U(1)) for 2D CFT.

    The anomaly is the obstruction to gauging G.
    """
    # Extract anomaly from partition function on non-trivial G-bundles
    # Relate to central charge and conformal weights via modular data

    # For abelian G, use Smith normal form
    # For non-abelian G, use group cohomology spectral sequence

    return CohomologyElement(...)
```

Test Cases:

- Detect symmetry in Ising model (c=1/2)
- Compute anomaly for SU(2) Wess-Zumino-Witten model
- Verify triviality of anomaly after gauging

0.3.4 Phase 4: Cobordism Invariants (Months 6-8)

Implement bordism group calculations:

```
from sympy.topology import simplicial_complex

class BordismElement:
    """Element of bordism group   ^{Spin}_d."""

    def __init__(self, dimension: int, characteristic_classes: dict):
        self.dim = dimension
        self.classes = characteristic_classes

    def arf_invariant(self) -> int:
        """Compute Arf invariant for d=1 (  ^{Spin}_1 =   _2  )."""
        if self.dim != 1:
            raise ValueError("Arf invariant only defined in d=1")
        # Use quadratic form on H_1
        return self.classes.get('arf', 0)

    def signature(self) -> int:
        """For d=4k, signature/8 mod 2 detects   ^{Spin}_{4k} =    ."""
        if self.dim % 4 != 0:
            return 0
        return self.classes.get('signature', 0)

def compute_bordism_invariant(cft_data: CFTData) -> BordismElement:
    """
    Extract bordism invariant from CFT partition function.

    For d=2:   ^{Spin}_2 =   _2   detected by gravitational anomaly c
        mod 24
    """
    c = cft_data.c
    c_bar = cft_data.c_bar

    # Gravitational anomaly: c - c   must be divisible by 24
    grav_anomaly = (c - c_bar) % 24

    if grav_anomaly != 0:
        # Non-trivial element of   ^{Spin}_2
        return BordismElement(2, {'anomaly': int(grav_anomaly)})

    return BordismElement(2, {'anomaly': 0})

def check_trivialization_in_gravity(bordism_elem: BordismElement) ->
    bool:
    """
    Check if bordism obstruction is trivialized when coupled to gravity.

    In pure gravity (no matter), certain bordism groups become trivial.
    """
    if bordism_elem.dim == 2:
        # In 2D quantum gravity, c - c   = 0 required (no gravitational
            anomaly)
        return bordism_elem.classes.get('anomaly', 0) == 0

    # Implement checks for higher dimensions
    return True
```

Validation:

- Verify $^{Spin}2 = 2 via c mod 24$

- Check consistency for (2,2) superconformal theories (c = c)
- Test non-compact bosonization (c=1)

0.3.5 Phase 5: Integrated Swampland Checker (Months 8-10)

Combine all checks into a unified verification tool:

```
def swampland_check(cft_data: CFTData) -> SwamplandCertificate:
    """
    Comprehensive swampland consistency check.

    Returns certificate documenting all checks and violations.
    """
    cert = SwamplandCertificate(is_consistent=True, violations=[])

    # Check 1: Modular invariance
    cert.s_matrix = extract_s_matrix(cft_data.characters, cft_data.c)
    cert.s_unitarity_error = verify_s_matrix_unitarity(cert.s_matrix)

    if cert.s_unitarity_error != Rational(0):
        cert.is_consistent = False
        cert.violations.append(f"S-matrix not unitary: error =
            {cert.s_unitarity_error}")

    # Check 2: Global symmetries
    if cft_data.symmetry_group:
        G = cft_data.symmetry_group
        cert.anomaly_class = compute_anomaly(G, cft_data)

        if not cert.anomaly_class.is_trivial():
            # Non-trivial anomaly: must gauge or break
            cert.violations.append(
                f"Exact global symmetry {G} with non-trivial anomaly "
                f"{cert.anomaly_class} forbidden in quantum gravity"
            )
            cert.is_consistent = False

    # Check 3: Bordism obstructions
    cert.bordism_class = compute_bordism_invariant(cft_data)
```

```python

    if not check_trivialization_in_gravity(cert.bordism_class):
        cert.is_consistent = False
        cert.violations.append(
            f"Non-trivial bordism class {cert.bordism_class} not trivialized by gravity"
        )

    # Check 4: Spectrum positivity
    cert.spectrum = extract_spectrum(cft_data)
    cert.min_gap = compute_min_gap(cert.spectrum)

    h_min = min(h for h, h_bar, deg in cert.spectrum if h > 0)
    if h_min > cft_data.c / 12:
        cert.violations.append(f"Spectral gap h_min = {h_min} > c/12 = {cft_data.c/12}")
        # This is a warning, not necessarily a hard constraint

    # Check 5: Cardy growth
    if not verify_cardy_growth(cert.spectrum, cft_data.c):
        cert.violations.append("Spectrum does not exhibit Cardy growth")
        cert.is_consistent = False

    return cert

def extract_spectrum(cft_data: CFTData) -> List[Tuple[Fraction, Fraction, int]]:
    """Extract primary spectrum from partition function."""
    spectrum = []

    # Parse partition matrix to identify primaries
    for i in range(len(cft_data.characters)):
        for j in range(len(cft_data.characters_bar)):
            if cft_data.partition_matrix[i, j] > 0:
                h_i = extract_conformal_weight(cft_data.characters[i])
                h_bar_j = extract_conformal_weight(cft_data.characters_bar[j])
                deg = cft_data.partition_matrix[i, j]
                spectrum.append((h_i, h_bar_j, deg))

    return spectrum
```

0.3.6 Phase 6: Database Generation and Publication (Months 10-12)

Generate complete catalogs of gravity-compatible CFTs:

```python
def enumerate_rational_cfts(c_max: Rational,
                            max_primaries: int = 10) -> List[CFTData]:
    """
    Enumerate all rational CFTs with c     c_max passing swampland checks.
    """
    candidates = []

    # Iterate over rational central charges
    for c_num in range(1, int(c_max * 24) + 1):
        c = Rational(c_num, 24)

        # Generate candidate partition functions
        # Use modular invariance to constrain search
        for partition_matrix in generate_modular_invariant_partitions(c, max_primaries):
            cft_data = CFTData(
                c=c, c_bar=c,
                characters=virasoro_characters_at_c(c),
                characters_bar=virasoro_characters_at_c(c),
                partition_matrix=partition_matrix,
                symmetry_group=detect_global_symmetry(partition_matrix)
            )

            cert = swampland_check(cft_data)
            if cert.is_consistent:
                candidates.append(cft_data)

    return candidates

def export_database(cfts: List[CFTData], output_path: Path):
    """Export to JSON with exact arithmetic."""
    import json

    data = {
        'timestamp': datetime.now().isoformat(),
        'count': len(cfts),
        'cfts': [
            {
                'c': f"{cft.c.p}/{cft.c.q}",
                'num_primaries': len(cft.characters),
                's_matrix': [[str(x) for x in row] for row in cft.s_matrix.tolist()],
                'spectrum': [(f"{h.p}/{h.q}", f"{hb.p}/{hb.q}", d)
                            for h, hb, d in extract_spectrum(cft)]
            }
            for cft in cfts
        ]
    }

    output_path.write_text(json.dumps(data, indent=2))
```

0.4 4. Example Starting Prompt

```
You are a theoretical physicist specializing in quantum gravity and conformal field theory.
Your task is to implement a swampland consistency checker for 2D CFTs using modular
invariance, higher-form symmetries, and cobordism obstructions.

OBJECTIVE: Determine which rational CFTs at central charge c    2 are consistent with
quantum gravity.
```

```

PHASE 1 (Months 1-2): Build modular forms library
- Implement exact Virasoro character computations using q-series
- Code Dedekind eta function and modular transformations
- Validate against minimal models (c = 1 - 6/(m(m+1)) for m = 3,4,5)

PHASE 2 (Months 2-4): Extract modular S-matrices
- Implement S-transformation (        -1/  ) for characters
- Use PSLQ to recognize exact algebraic S-matrix entries
- Verify unitarity: ||S  S    -      || = 0 using exact arithmetic
- Check modular identities: (ST)    = S

PHASE 3 (Months 4-6): Analyze higher-form symmetries
- Detect global symmetry groups from partition function structure
- Implement group cohomology computations for H  (BG, U(1))
- Compute 't Hooft anomalies and check gauging obstructions

PHASE 4 (Months 6-8): Compute cobordism invariants
- Calculate gravitational anomaly: (c -  c  ) mod 24
- Determine bordism class in   ^{Spin}_2 =   _2
- Verify trivialization when coupled to gravity

PHASE 5 (Months 8-10): Integrate all checks
- Build unified swampland_check(cft_data) function
- Generate SwamplandCertificate with all verification data
- Export certificates as JSON with exact arithmetic

PHASE 6 (Months 10-12): Generate databases
- Enumerate all rational CFTs with c     2
- Apply swampland checks to filter gravity-compatible theories
- Export complete database with S-matrices and spectra

SUCCESS CRITERIA:
- MVR: Successfully reproduce S-matrices for minimal models, verify
    unitarity
- Strong: Detect and classify global symmetries, compute anomalies for
    c     1
- Publication: Complete database of gravity-compatible CFTs at c     2
    with certificates

VERIFICATION:
- All S-matrices verified unitary using exact arithmetic (error = 0)
- All anomalies and bordism invariants computed symbolically
- Certificates exported as JSON with rational/algebraic numbers
- Results cross-checked against known CFT classifications

Use only symbolic computation (sympy, mpmath with 100+ digit
    precision). Generate
machine-checkable certificates for all consistency checks.
```

0.5 5. Success Criteria

0.5.1 Minimum Viable Result (MVR)

Within 2-4 months, the system should:

- **Modular Bootstrap Library**:
- Compute Virasoro characters for c 1/2, 7/10, 4/5, 1 (minimal models)
- Extract exact S-matrices for Ising (c=1/2), 3-state Potts (c=4/5)
- Verify S-unitarity with error = 0 using rational arithmetic
- **Basic Swampland Checks**:
- Detect symmetry in Ising model
- Verify (c - c) mod 24 = 0 for diagonal theories
- Flag theories with exact global symmetries
- **Validation**:
- Reproduce 5+ known minimal model S-matrices from literature
- All unitarity checks pass with exact zero error

Deliverable: `swampland`$_{m}inimal.pywithbasicchecks, JSONoutputfor5validatedCFTs$

0.5.2 Strong Result

Within 6-8 months, add:

- **Higher-Form Symmetry Analysis**:
- Compute H^3(BG, U(1)) for cyclic groups G = (n 12)
- Detect and classify anomalies in all minimal models (c < 1)
- Verify anomaly trivialization for gauged theories (e.g., orbifolds)
- **Extended Database**:
- Enumerate all rational CFTs with c 1.5
- Apply full swampland checks to 50+ candidate theories
- Export database with S-matrices, spectra, anomaly classes
- **Cobordism Calculations**:
- Compute Arf invariants for all minimal models
- Verify gravitational anomaly constraints for (2,2) SCFTs
- Implement checks for non-diagonal modular invariants

Metrics:

- Database contains 50+ verified gravity-compatible CFTs
- All anomaly classes computed exactly (symbolic cohomology)
- Cross-validation: reproduce Gaiotto-Johnson-Freyd classification results

0.5.3 Publication-Quality Result

Within 9-12 months, achieve:

- **Complete c 2 Classification**:
- Enumerate all modular-invariant partition functions at c 2
- Apply swampland checks to 200+ candidates
- Identify novel gravity-compatible CFTs not in existing literature
- **Non-Abelian Symmetries**:
- Extend anomaly calculations to non-abelian groups (A, S, etc.)
- Compute anomalies for WZW models at low levels
- Verify completeness hypothesis for finite gauge groups
- **Higher-Dimensional Extensions**:
- Generalize to 3D CFTs via F-maximization and a-maximization
- Implement 4D cobordism checks ($_4^{Spin} =$)

- Connect to EFT positivity bounds in d > 2
- **Formal Verification**:
- Translate key theorems to Lean/Isabelle
- Formally verify S-matrix unitarity and modular identities
- Generate computer-checkable proofs of swampland constraints

Publication Potential:

- "Computational Classification of Gravity-Compatible 2D CFTs"
- "Modular Bootstrap Meets the Swampland: Exact Results at c 2"
- "Automated Swampland Checks via Higher-Form Symmetries"

Impact: Provides first complete, computer-verified catalog of 2D CFTs consistent with quantum gravity.

0.6 6. Verification Protocol

0.6.1 Automated Checks

```python
def verify_swampland_certificate(cert: SwamplandCertificate) -> bool:
    """
    Verify all claims in swampland certificate using independent checks.
    """
    checks_passed = []

    # Check 1: S-matrix unitarity
    S = cert.s_matrix
    unitarity_check = (S * S.conjugate().transpose() ==
        Matrix.eye(S.rows))
    checks_passed.append(('S-unitarity', unitarity_check))

    # Check 2: Modular identities
    # Compute T-matrix from character transformations          +1
    T = extract_t_matrix(cert)
    st_cubed_check = ((S * T)**3 == S**2)
    checks_passed.append(('(ST)^3 = S^2', st_cubed_check))

    # Check 3: Anomaly calculation
    if cert.anomaly_class:
        # Recompute anomaly independently
        anomaly_recomputed = compute_anomaly_independent(cert)
        checks_passed.append(('Anomaly', cert.anomaly_class ==
            anomaly_recomputed))

    # Check 4: Bordism invariant
    bordism_recomputed =
        compute_bordism_invariant_from_spectrum(cert.spectrum)
    checks_passed.append(('Bordism', cert.bordism_class ==
        bordism_recomputed))

    # Check 5: Cardy formula
    if cert.spectrum:
        cardy_check = verify_cardy_asymptotic(cert.spectrum, cert.c)
        checks_passed.append(('Cardy growth', cardy_check))

    print("Verification Results:")
    for check_name, passed in checks_passed:
        status = "    PASS" if passed else "    FAIL"
        print(f"  {status}: {check_name}")

    return all(passed for _, passed in checks_passed)
```

0.6.2 Cross-Validation

Compare against known results:

- **Minimal Models:** ADE classification (Cappelli-Itzykson-Zuber)
- **WZW Models:** Kac-Moody S-matrices (Kac-Peterson)
- **Orbifolds:** Check against Dixon-Ginsparg-Harvey results

- **Swampland Literature:** Verify constraints from McNamara-Vafa, Ooguri-Vafa

0.6.3 Exported Artifacts

For each verified CFT:

- **Certificate JSON:**

```
{
  "cft_id": "minimal_m3_Ising",
  "c": "1/2",
  "num_primaries": 3,
  "s_matrix": [
    ["1/2", "1/2", "1/sqrt(2)"],
    ["1/2", "1/2", "-1/sqrt(2)"],
    ["1/sqrt(2)", "-1/sqrt(2)", "0"]
  ],
  "unitarity_error": "0",
  "symmetry_group": "Z2",
  "anomaly_class": "trivial",
  "bordism_class": "0 in Z2",
  "spectrum": [
    {"h": "0", "h_bar": "0", "degeneracy": 1, "label": "vacuum"},
    {"h": "1/16", "h_bar": "1/16", "degeneracy": 1, "label":
        "sigma"},
    {"h": "1/2", "h_bar": "1/2", "degeneracy": 1, "label":
        "epsilon"}
  ],
  "is_consistent": true,
  "violations": []
}
```

- **S-Matrix Verification Certificate** (symbolic proof of unitarity)
- **Cohomology Computation Log** (spectral sequence pages for anomaly)
- **Comparison Report** (vs. known literature results)

0.7 7. Resources Milestones

0.7.1 Key References

Modular Bootstrap:

- Cardy (1986): "Operator Content of Two-Dimensional Conformally Invariant Theories"
- Cappelli, Itzykson, Zuber (1987): "Modular Invariant Partition Functions"
- Friedan, Shenker (2024): "2D Modular Bootstrap"

Higher-Form Symmetries:

- Gaiotto, Kapustin, Seiberg, Willett (2015): "Generalized Global Symmetries"
- Córdova, Dumitrescu, Intriligator (2019): "Exploring 2-Group Global Symmetries"

Swampland Program:

- Vafa (2005): "The String Landscape and the Swampland"
- Ooguri, Vafa (2006): "Non-supersymmetric AdS and the Swampland"
- McNamara, Vafa (2019): "Cobordism Classes and the Swampland"

Cobordism and TFT:

- Freed, Hopkins (2021): "Reflection Positivity and Invertible Topological Phases"
- Johnson-Freyd (2020): "Topological Mathieu Moonshine"

0.7.2 Common Pitfalls

- **Numerical Precision in Modular Transformations:**
- Problem: S-matrix extraction requires high-precision evaluation
- Solution: Use mpmath with 100+ digits, PSLQ for exact recognition
- **Group Cohomology for Large Groups:**
- Problem: H^3(BG, U(1)) is hard to compute for non-abelian G
- Solution: Use Lyndon-Hochschild-Serre spectral sequence, implement in GAP
- **Partition Function Enumeration:**
- Problem: Exponentially many candidate partition matrices
- Solution: Use modular invariance to prune search space, SAT solver for constraints
- **Spectral Sequence Convergence:**
- Problem: Cohomology spectral sequences may not stabilize quickly
- Solution: Bound spectral sequence pages using representation theory

0.7.3 Milestone Checklist

Month 2:	Modular forms library complete, 5 minimal models validated
Month 4:	S-matrix extraction working, unitarity verified for 10+ CFTs
Month 6:	Anomaly calculations for cyclic groups, database of 20 CFTs
Month 8:	Cobordism invariants computed, borderline cases identified
Month 10:	Integrated swampland checker, 50+ CFTs verified
Month 12:	Complete c 2 classification, publication draft ready

---

0.8 8. Extensions and Open Questions

0.8.1 Immediate Extensions

- **3D CFTs and F-Maximization:** Extend swampland checks to supersymmetric 3D CFTs using exact results from localization

- **4D EFT Positivity Bounds**: Connect CFT swampland constraints to Wilson coefficient bounds in 4D effective field theories
- **Machine Learning for Partition Function Search**: Train models to predict modular-invari partition matrices

0.8.2 Research Frontiers

- **Non-Rational CFTs**: Can we extend swampland checks to non-rational CFTs (irrational central charge)?
- **Holographic Duality**: What is the AdS gravity dual interpretation of swampland constraints?
- **Quantum Error Correction**: Do gravity-compatible CFTs have special properties as quantum codes?

0.8.3 Long-Term Vision

Build a **Swampland Database** covering:
- 2D CFTs (rational and non-rational)
- 3D SCFTs
- 4D N=2 SCFTs
- 6D (2,0) theories

All verified by automated modular bootstrap and higher-form symmetry checks, providing a computational foundation for the swampland program.

End of PRD 08