PRD 22: Bell Inequalities and Quantum Nonlocality

Pure Thought AI Challenge 22

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 Information Theory
Timeline: 6-9 months
Difficulty: High
Prerequisites: Quantum mechanics, probability theory, convex optimization, SDP solvers, linear algebra

---

0.1 1. Problem Statement

0.1.1 Scientific Context

Bell's theorem (1964) stands as one of the most profound results in 20th-century physics, demonstrating that quantum mechanics is fundamentally incompatible with local realism—the assumption that physical properties exist independently of measurement and that influences cannot propagate faster than light. Bell inequalities provide mathematical constraints that any local hidden variable (LHV) theory must satisfy, while quantum mechanics predicts violations of these constraints for entangled states. Experimental tests by Aspect, Clauser, and Zeilinger (Nobel Prize 2022) confirmed quantum violations, ruling out entire classes of classical explanations.

Bell inequalities are linear functionals on the space of probability distributions arising from joint measurements by spatially separated parties. The **CHSH inequality** (Clauser-Horne-Shimony-Holt, 1969) for two parties with binary measurements states: S = E(AB) + E(AB) + E(AB) - E(AB) 2 for all LHV theories, where E(AB) are correlations. Quantum mechanics achieves S = 22 2.828 (Tsirelson's bound, 1980) using maximally entangled states and appropriate measurements. This "quantum advantage" has profound implications for cryptography (device-independent quantum key distribution), randomness generation, and tests of quantum foundations.

The **NPA hierarchy** (Navarro-Pironio-Acín, 2007-2008) provides a systematic method to compute quantum bounds on Bell violations by solving a sequence of semidefinite programs (SDPs) that converge to the quantum set Q. At level k, the NPA hierarchy imposes consistency constraints on k-body correlators ⟨A*i*...*A*i⟩ derived from the operator structure of quantum mechanics. **Facet enumeration** of the local polytope (via vertex enumeration and duality) yields all possible Bell inequalities for a given scenario. **Device-independent certification** leverages Bell violations to certify properties like randomness and entanglement without assumptions about the internal workings of measurement devices.

0.1.2 Core Question

Given a Bell scenario (N, M, d) with N parties, M measurements per party, and d outcomes each:

- Enumerate all tight Bell inequalities (facets of the local polytope)
- Compute quantum bounds (Tsirelson bounds) via NPA hierarchy
- Find optimal quantum states and measurement bases achieving maximal violations
- Certify device-independent randomness and entanglement using observed correlations
- Generate machine-checkable certificates for all quantum bounds and inequality derivations

0.1.3 Why This Matters

- **Foundational Physics**: Bell inequalities provide experimental tests distinguishing quantum mechanics from classical theories, resolving century-old debates about quantum foundations (EPR paradox, 1935)
- **Device-Independent Cryptography**: Bell violations enable cryptographic protocols secure against adversaries controlling measurement devices (DIQKD protocols certified by CHSH > 2)
- **Quantum Networking**: Certification of entanglement distribution in quantum networks without trusting intermediate nodes
- **Randomness Expansion**: Guaranteed unpredictable random bits from Bell violations, crucial for cryptography and Monte Carlo simulations
- **Theoretical Boundaries**: Understanding the "shape" of quantum correlations reveals fundamental structure of nature (Q lies strictly between L and NS, the no-signaling set)

0.1.4 Pure Thought Advantages

- **Certificate-Based Verification**: All Bell inequalities and quantum bounds come with SDP certificates (dual feasibility proofs) and vertex enumeration records
- **Exact Arithmetic**: Use `sympy` for rational coefficients in Bell inequalities, ensuring symbolic correctness of facet equations
- **Convergence Guarantees**: NPA hierarchy converges to Q; can bound approximation error at each level
- **No Experimental Noise**: Pure thought analysis avoids detector inefficiencies and measurement imperfections that complicate real experiments (closing loopholes)
- **Completeness**: Facet enumeration via convex hull algorithms guarantees all Bell inequalities are found for small scenarios

---

0.2 2. Mathematical Formulation

0.2.1 Bell Scenario (N, M, d)

A **Bell scenario** is specified by:

- **N**: Number of parties (typically 2 for bipartite)
- **M**: Number of measurement settings per party
- **d**: Number of outcomes per measurement

Each party chooses measurement x 0, ..., M-1 and observes outcome a 0, ..., d-1. The correlation is a probability distribution:

P(a...a | x...x) **[0,1]** with $_{a...a}P(a...a|x...x) = 1$

0.2.2 Local Hidden Variable (LHV) Model

A distribution P is **local** if there exists:

P(a...a | x...x) = $_{p()P(a|x,)P(a|x,)...P(a|x,)}$

where is a shared hidden variable with prior p(), and each party's response depends only on their local measurement choice x and (no faster-than-light influence).

The set of all local distributions forms a polytope L (convex hull of deterministic strategies). A deterministic strategy assigns definite outcomes: a = f(x,) for each (x,).

0.2.3 Quantum Model

A distribution P is quantum if there exists:

- **A Hilbert space H = H ... H**
- **A state |⟩ H (or density matrix)**
- **POVM elements M for each party (positive operators summing to identity)**

Such that:

P(a...a | x...x) = ⟨| M^{x}

0.2.4 Bell Inequality

A **Bell inequality** is a linear functional : P → and bound $_{L} such that$:

(P) $_{L} for all P L$

A violation occurs when (P $Q)$ **> L for some quantum P**$_{Q}Q$.

Example (CHSH): For (2,2,2) scenario, the CHSH functional is:

S = ⟨AB⟩ + ⟨AB⟩ + ⟨AB⟩ - ⟨AB⟩

where ⟨AB⟩ = $_{a,b}(-1)^{a+b}P(ab|ij)$.

- **Local bound: S 2**
- **Tsirelson bound: S 22 (quantum maximum)**
- **No-signaling bound: S 4**

0.2.5 NPA Hierarchy

The NPA (Navarro-Pironio-Acín) hierarchy constructs a sequence of SDP relaxations:

Q Q ... Q converging to the quantum set Q.

At level k, construct the moment matrix indexed by operator sequences of length k:

$_{S,T} = <|S\check{D}T|>$

where S, T are products of measurement operators. Constraints:

- **Positive semidefinite: 0**
- **Normalization:** $_{,} = 1$

- **Hermiticity:** S,T **= *T,S**
- **Commutation: [A, B] = 0 for spacelike-separated operators**

- **POVM:** $\boldsymbol{a}$ $M^x a = I foreachmeasurementx$

  Convergence: As k → , Q → Q (Doherty-Parrilo-Spedalieri, 2008).

0.2.6 Certificate Specification

A Bell inequality certificate must contain:

  - **Facet representation: Coefficients** $= (_{a\ldots a,x\ldots x}) asexactrationals$

- **Vertex witness: Set of dim(L) affinely independent deterministic strategies saturating (P) =** $_L$
- **Local bound:** $_L computedasmaxoververtices$
- **Quantum bound:** $_Q fromNPAhierarchywithSDPsolvercertificate(dualoptimum, primalfeasibility)$
- **Optimal strategy: Quantum state and measurements M**^{x} $\boldsymbol{a} achieving(PQ) =_Q (uptonumericaltoleranc$ 10ź)
- **No-signaling check: Verify all marginals satisfy non-signaling constraints**

0.3 3. Implementation Approach

This is a 6-phase project spanning 6-9 months, building from simple 2-party scenarios to general N-party inequalities and device-independent protocols.

0.3.1 Phase 1: CHSH and Simple Bell Inequalities (Months 1-2)

Objective: Implement CHSH inequality, compute local and quantum bounds analytically.

```python
import numpy as np
import sympy as sp
from dataclasses import dataclass
from typing import List, Dict, Tuple

@dataclass
class BellScenario:
    """Bell scenario specification."""
    N: int  # Number of parties
    M: int  # Measurements per party
    d: int  # Outcomes per measurement

    def dim_probability_space(self) -> int:
        """Dimension of probability distribution P( a  ... a   |
            x  ... x  )."""
        return self.M**self.N * self.d**self.N

    def dim_local_polytope(self) -> int:
        """Dimension of local polytope L."""
        # Each party has d^M deterministic strategies
        return self.N * (self.d**self.M - 1)  # Subtract 1 for
            normalization

```

```
def chsh_correlator(P: np.ndarray) -> float:
    """
    Compute CHSH correlator S for (2,2,2) scenario.

    P[a,b,x,y] = P(ab|xy) for outcomes a,b     {0,1}, measurements x,y     {0,1}

    S = E(  A   B   ) + E(  A   B   ) + E(  A   B   ) - E(  A   B   )
    where E(  A   B   ) =   _ {a,b} (-1)^{a+b} P(ab|xy)

    Local bound: S     2
    Tsirelson bound: S     2  2
    """
    def E(x: int, y: int) -> float:
        corr = 0.0
        for a in [0, 1]:
            for b in [0, 1]:
                corr += (-1)**(a+b) * P[a, b, x, y]
        return corr

    S = E(0, 0) + E(0, 1) + E(1, 0) - E(1, 1)
    return S

def chsh_quantum_optimal() -> Dict:
    """
    Compute optimal quantum strategy for CHSH.

    Returns: state, measurements, and maximal violation 2  2  .
    """
    # Optimal state: singlet |          = (|01     - |10    )/  2
    psi = np.array([0, 1, -1, 0], dtype=complex) / np.sqrt(2)

    # Optimal measurements: Pauli operators with angles
    # Alice: A    =  _z ,  A    =  _x
    # Bob: B    = ( _z   +  _x )/  2  ,  B    = ( _z   -  _x )/  2

    theta_A0 = 0.0
    theta_A1 = np.pi/2
    theta_B0 = np.pi/4
    theta_B1 = -np.pi/4

    def measurement_operator(theta: float) -> np.ndarray:
        """Projective measurement along axis at angle theta in XZ plane."""
        return np.array([[np.cos(theta), np.sin(theta)],
                         [np.sin(theta), -np.cos(theta)]])

    A = [measurement_operator(theta_A0), measurement_operator(theta_A1)]
    B = [measurement_operator(theta_B0), measurement_operator(theta_B1)]

    # Compute correlations
    P = np.zeros((2, 2, 2, 2))
    for x in [0, 1]:
        for y in [0, 1]:
            for a in [0, 1]:
                for b in [0, 1]:
```

```
                    # POVM projectors (for dichotomic  1 outcomes, map to 0/1)
                    M_a = np.kron(A[x] if a == 0 else np.eye(2) - A[x], np.eye(2))
                    M_b = np.kron(np.eye(2), B[y] if b == 0 else np.eye(2) - B[y])
                    P[a, b, x, y] = np.real(psi.conj() @ M_a @ M_b @ psi)

    S_value = chsh_correlator(P)

    return {
        'state': psi,
        'measurements_A': A,
        'measurements_B': B,
        'correlations': P,
        'chsh_value': S_value,
        'tsirelson_bound': 2 * np.sqrt(2)
    }

def verify_chsh_local_bound() -> Dict:
    """
    Verify CHSH local bound S   2 by checking all deterministic strategies.

    For (2,2,2): each party has 2  = 4 deterministic strategies.
    Total: 4   4 = 16 product strategies.
    """
    max_S = -np.inf
    worst_strategy = None

    # Deterministic strategy: a = f(x)   {0,1} for x   {0,1}
    for f_A in range(4):  # 00, 01, 10, 11 (outputs for x=0,1)
        for f_B in range(4):
            # Decode deterministic responses
            a0 = (f_A >> 1) & 1
            a1 = f_A & 1
            b0 = (f_B >> 1) & 1
            b1 = f_B & 1

            # Build deterministic correlation
            P = np.zeros((2, 2, 2, 2))
            P[a0, b0, 0, 0] = 1.0
            P[a0, b1, 0, 1] = 1.0
            P[a1, b0, 1, 0] = 1.0
            P[a1, b1, 1, 1] = 1.0

            S = chsh_correlator(P)
            if S > max_S:
                max_S = S
                worst_strategy = (f_A, f_B)

    return {
        'local_bound': max_S,
        'is_exactly_2': np.isclose(max_S, 2.0),
        'worst_strategy': worst_strategy
```

```python
    }

# Example usage
if __name__ == "__main__":
    # Verify local bound
    local_result = verify_chsh_local_bound()
    print(f"CHSH local bound: {local_result['local_bound']:.6f} (should be 2)")

    # Compute quantum optimal
    quantum_result = chsh_quantum_optimal()
    print(f"CHSH quantum value: {quantum_result['chsh_value']:.6f}")
    print(f"Tsirelson bound: {quantum_result['tsirelson_bound']:.6f}")
    print(f"Violation: {quantum_result['chsh_value'] - 2.0:.6f}")
```

0.3.2 Phase 2: NPA Hierarchy for Quantum Bounds (Months 2-4)

Objective: Implement NPA hierarchy at levels 1, 2, 3 using SDP solvers (CVXPY + MOSEK).

```python
import cvxpy as cp
from itertools import product, combinations_with_replacement

def generate_npa_moments(scenario: BellScenario, level: int) -> List[Tuple]:
    """
    Generate operator sequences for NPA hierarchy at given level.

    Level 1: {I, A^x_a, B^y_b, A^x_a B^y_b}
    Level k: All products of     k measurement operators

    Returns: List of operator sequences as tuples (party, measurement, outcome).
    """
    moments = [()]  # Identity

    # Single operators
    for party in range(scenario.N):
        for x in range(scenario.M):
            for a in range(scenario.d):
                moments.append(((party, x, a),))

    # Higher-level products
    if level >= 2:
        # Two-body correlators
        for op1 in moments[1:]:  # Skip identity
            for op2 in moments[1:]:
                if len(op1) + len(op2) <= level:
                    moments.append(op1 + op2)

    # Remove duplicates (order matters for non-commuting operators)
    # Implement commutation rules: [A^x_a, B^y_b] = 0 for different parties
    moments = list(set(moments))
    return moments
```

```
def npa_sdp_chsh(level: int = 1) -> Dict:
    """
    NPA SDP for CHSH inequality at given level.

    Maximize S =       AB       +       AB       +       AB       -       AB

    Returns: optimal quantum bound and certificate.
    """
    scenario = BellScenario(N=2, M=2, d=2)
    moments = generate_npa_moments(scenario, level)
    n_moments = len(moments)

    # Create moment matrix    (PSD variable)
    Gamma = cp.Variable((n_moments, n_moments), PSD=True)

    # Index mapping: moment sequence     matrix index
    moment_to_idx = {m: i for i, m in enumerate(moments)}

    constraints = []

    # Normalization:   [I, I] = 1
    constraints.append(Gamma[0, 0] == 1)

    # POVM constraints:  _a  A^x_a = I
    for party in range(scenario.N):
        for x in range(scenario.M):
            povm_sum = 0
            for a in range(scenario.d):
                idx = moment_to_idx.get(((party, x, a),))
                if idx is not None:
                    povm_sum += Gamma[0, idx]  #   I  , A^ x _ a   = Tr( A^x_a)
            constraints.append(povm_sum == 1)

    # Commutativity: [A, B] = 0         AB     =     BA
    # For different parties, operators commute
    for m1 in moments:
        for m2 in moments:
            if len(m1) > 0 and len(m2) > 0:
                party1 = m1[-1][0] if len(m1) > 0 else None
                party2 = m2[0][0] if len(m2) > 0 else None
                if party1 != party2:
                    #   [m1 m2] should equal   [m2 m1]
                    seq_12 = m1 + m2
                    seq_21 = m2 + m1
                    idx_12 = moment_to_idx.get(seq_12)
                    idx_21 = moment_to_idx.get(seq_21)
                    if idx_12 is not None and idx_21 is not None:
                        constraints.append(Gamma[0, idx_12] == Gamma[0, idx_21])

    # Objective: maximize CHSH correlator
    # S = E(  A  B  ) + E(  A  B  ) + E(  A  B  ) - E(  A  B  )
    # E(  A  B  ) =  _ {a,b} (-1)^{a+b}   A  ^x_a B^ y _ b
```

```python

    S_expr = 0
    for x, y, sign in [(0, 0, 1), (0, 1, 1), (1, 0, 1), (1, 1, -1)]:
        for a in range(2):
            for b in range(2):
                coeff = sign * (-1)**(a + b)
                # Find moment for A^x_a B^y_b
                seq = ((0, x, a), (1, y, b))  # Party 0 (Alice), Party
                    1 (Bob)
                idx = moment_to_idx.get(seq)
                if idx is not None:
                    S_expr += coeff * Gamma[0, idx]

    # Solve SDP
    problem = cp.Problem(cp.Maximize(S_expr), constraints)
    problem.solve(solver=cp.MOSEK, verbose=False)

    return {
        'quantum_bound': problem.value,
        'level': level,
        'status': problem.status,
        'dual_certificate': problem.constraints,
        'moments': moments
    }

# Example: Compute NPA bounds at different levels
for level in [1, 2]:
    result = npa_sdp_chsh(level)
    print(f"NPA level {level}: S_max = {result['quantum_bound']:.8f}")
    print(f"  (Tsirelson = {2*np.sqrt(2):.8f})")
```

0.3.3 Phase 3: Facet Enumeration of Local Polytope (Months 4-5)

Objective: Enumerate all Bell inequality facets for small scenarios via vertex enumeration.

```python
from scipy.spatial import ConvexHull
import sympy as sp

def generate_deterministic_strategies(scenario: BellScenario) ->
    np.ndarray:
    """
    Generate all deterministic strategies (vertices of local polytope).

    Each party has d^M deterministic strategies: f: {0,...,M-1}
        {0,...,d-1}
    Total: (d^M)^N product strategies.

    Returns: Array of shape (n_vertices, dim_probability_space)
    """
    n_strategies_per_party = scenario.d ** scenario.M
    n_vertices = n_strategies_per_party ** scenario.N

    # Probability space dimension
    dim_prob = scenario.M**scenario.N * scenario.d**scenario.N
```

```python

    vertices = []

    # Iterate over all product strategies
    for strategy_indices in product(range(n_strategies_per_party),
        repeat=scenario.N):
        # Decode each party's deterministic function
        functions = []
        for party_idx, strat_idx in enumerate(strategy_indices):
            # strat_idx encodes function f: {0,...,M-1}     {0,...,d-1}
            # Treat as base-d number with M digits
            f = {}
            temp = strat_idx
            for x in range(scenario.M):
                f[x] = temp % scenario.d
                temp //= scenario.d
            functions.append(f)

        # Build probability distribution P( a  ... a   |  x  ... x  )
        P = np.zeros(dim_prob)
        idx = 0
        for inputs in product(range(scenario.M), repeat=scenario.N):
            for outputs in product(range(scenario.d),
                repeat=scenario.N):
                # Check if this output matches the deterministic
                    functions
                match = True
                for party in range(scenario.N):
                    if outputs[party] !=
                        functions[party][inputs[party]]:
                        match = False
                        break
                P[idx] = 1.0 if match else 0.0
                idx += 1

        vertices.append(P)

    return np.array(vertices)

def enumerate_bell_inequalities(scenario: BellScenario) -> List[Dict]:
    """
    Enumerate all facets of local polytope L (i.e., all tight Bell
        inequalities).

    Uses convex hull algorithm on deterministic strategies.

    Returns: List of Bell inequalities as {coefficients, bound,
        vertices}.
    """
    vertices = generate_deterministic_strategies(scenario)

    # Compute convex hull
    hull = ConvexHull(vertices)

    # Each facet equation: n    x     b (normal n, offset b)
    inequalities = []
```

```
    for i, eq in enumerate(hull.equations):
        # eq = [ n  , n  , ..., n  , b] where n x + b     0
        # Rewrite as n x     -b
        coeffs = eq[:-1]
        bound = -eq[-1]

        # Find vertices on this facet
        facet_vertices = hull.simplices[i] if i < len(hull.simplices) else []

        # Convert to exact rationals using sympy
        coeffs_rational = [sp.Rational(c).limit_denominator(10**6) for c in coeffs]
        bound_rational = sp.Rational(bound).limit_denominator(10**6)

        inequalities.append({
            'coefficients': coeffs_rational,
            'bound': bound_rational,
            'facet_vertices': facet_vertices.tolist()
        })

    return inequalities

# Example: Enumerate all Bell inequalities for (2,2,2)
scenario_222 = BellScenario(N=2, M=2, d=2)
bell_inequalities = enumerate_bell_inequalities(scenario_222)
print(f"Found {len(bell_inequalities)} Bell inequalities for (2,2,2)")

# Identify CHSH among them
for i, ineq in enumerate(bell_inequalities):
    # Check if coefficients match CHSH pattern
    # CHSH has specific structure: 4 non-zero correlations with signs +1, +1, +1, -1
    print(f"\nInequality {i}:")
    print(f"  Bound: {ineq['bound']}")
    print(f"  Non-zero coeffs: {sum(1 for c in ineq['coefficients'] if c != 0)}")
```

0.3.4 Phase 4: Optimal Quantum Strategies (Months 5-6)

Objective: Given a Bell inequality, find optimal quantum state and measurements.

```
def optimize_quantum_strategy(bell_inequality: Dict, scenario: BellScenario,
                              dim_hilbert: int = 2) -> Dict:
    """
    Find optimal quantum strategy (state + measurements) for given Bell inequality.

    Uses NPA hierarchy + state/measurement extraction.

    Args:
        bell_inequality: Coefficients    and bound   _L
        scenario: Bell scenario (N, M, d)
        dim_hilbert: Hilbert space dimension per party

```

```
    Returns: Optimal state    , measurements {M^x_a}, and achieved value.
    """
    # Extract moments from NPA solution
    npa_result = npa_sdp_general(bell_inequality, scenario, level=2)
    Gamma_opt = npa_result['moment_matrix']

    # Perform eigendecomposition of Gamma
    eigvals, eigvecs = np.linalg.eigh(Gamma_opt)

    # Largest eigenvalue corresponds to state |
    idx_max = np.argmax(eigvals)
    psi_flat = eigvecs[:, idx_max]

    # Reshape to multipartite state (assumes product Hilbert space structure)
    dim_total = dim_hilbert ** scenario.N
    psi = psi_flat[:dim_total]
    psi /= np.linalg.norm(psi)

    # Extract measurements from moment matrix consistency
    #   A ^ x _ a   = Gamma[0, idx(A^x_a)] gives expectation value
    measurements = {}
    for party in range(scenario.N):
        measurements[party] = {}
        for x in range(scenario.M):
            # Reconstruct POVM {M^x_a} from moments
            # This is non-trivial; use SDP to enforce   A ^ x _ a   = Tr(   M^x_a)
            M_x = []
            for a in range(scenario.d):
                # Placeholder: extract from moment matrix (requires advanced techniques)
                M_a = np.eye(dim_hilbert) / scenario.d  # Uniform POVM as placeholder
                M_x.append(M_a)
            measurements[party][x] = M_x

    # Compute achieved Bell value
    beta_value = evaluate_bell_inequality(bell_inequality, psi, measurements, scenario)

    return {
        'state': psi,
        'measurements': measurements,
        'bell_value': beta_value,
        'violation': beta_value - float(bell_inequality['bound'])
    }

def evaluate_bell_inequality(bell_ineq: Dict, state: np.ndarray,
                             measurements: Dict, scenario: BellScenario) -> float:
    """
    Evaluate Bell inequality value   P   for given quantum strategy.

    Returns:  _ {a,x}  _ {a,x} P(a|x) where P computed from
          |M|     .
```

```
    """
    # Compute probability distribution P( a  ... a   |  x  ... x  )
    P = compute_quantum_correlations(state, measurements, scenario)

    # Contract with Bell inequality coefficients
    coeffs = bell_ineq['coefficients']
    value = np.dot(coeffs, P.flatten())

    return value

def compute_quantum_correlations(state: np.ndarray, measurements: Dict,
                                 scenario: BellScenario) -> np.ndarray:
    """
    Compute P( a  ... a   |  x  ... x  ) =      | M^{ x  }_{ a  }
       ...     M^{ x  }_{ a  } |      .
    """
    shape = tuple([scenario.d] * scenario.N + [scenario.M] * scenario.N)
    P = np.zeros(shape)

    for inputs in product(range(scenario.M), repeat=scenario.N):
        for outputs in product(range(scenario.d), repeat=scenario.N):
            # Tensor product of measurements
            M_joint = measurements[0][inputs[0]][outputs[0]]
            for party in range(1, scenario.N):
                M_joint = np.kron(M_joint,
                   measurements[party][inputs[party]][outputs[party]])

            # Expectation value
            P[outputs + inputs] = np.real(state.conj() @ M_joint @
               state)

    return P
```

0.3.5 Phase 5: Device-Independent Certification (Months 6-7)

Objective: Certify randomness, entanglement, and security from observed Bell violations.

```
def certify_device_independent_randomness(correlations: np.ndarray,
                                          scenario: BellScenario) ->
                                              Dict:
    """
    Certify device-independent randomness from observed correlations.

    Uses min-entropy bounds from NPA hierarchy.

    H_min(A|E)     - l o g  (p_guess) where p_guess is guessing
       probability.

    Returns: Min-entropy bits per measurement.
    """
    # Compute Bell violation
    chsh_value = chsh_correlator(correlations)

    if chsh_value <= 2.0:
```

```
        return {
            'randomness': 0.0,
            'chsh_value': chsh_value,
            'certified': False,
            'reason': 'No Bell violation'
        }

    # Use Ac n et al. (2012) bound for CHSH:
    # H_min( A  |E)     1 - h((1 +    ((S  /8) - 1))/2)
    # where h(x) = -x  log (x) - (1-x)  log (1-x) is binary entropy

    S = chsh_value

    def binary_entropy(x: float) -> float:
        if x <= 0 or x >= 1:
            return 0.0
        return -x * np.log2(x) - (1 - x) * np.log2(1 - x)

    if S**2 / 8 < 1:
        return {'randomness': 0.0, 'certified': False,
                'reason': 'Insufficient violation for randomness'}

    p_win = (1 + np.sqrt(S**2 / 8 - 1)) / 2
    H_min = 1 - binary_entropy(p_win)

    return {
        'randomness': H_min,
        'chsh_value': S,
        'certified': True,
        'guessing_probability': p_win,
        'bits_per_round': H_min
    }

def certify_entanglement_dimension(correlations: np.ndarray) -> Dict:
    """
    Certify minimum Schmidt rank of entangled state from Bell violation.

    Returns: Lower bound on Schmidt rank (dimension of entanglement).
    """
    # Use dimension witnesses (Brunner et al., PRL 2008)
    # Different Bell inequalities have different Schmidt rank requirements

    chsh_value = chsh_correlator(correlations)

    if chsh_value > 2.0:
        # CHSH violation requires at least Schmidt rank 2 (genuine entanglement)
        schmidt_rank_min = 2
    else:
        schmidt_rank_min = 1  # Separable

    # More sophisticated: use NPA hierarchy with rank constraints
    # Q^{(1+AB)} relaxation certifies Schmidt rank     2

    return {
```

```
        'min_schmidt_rank': schmidt_rank_min,
        'chsh_value': chsh_value,
        'entangled': chsh_value > 2.0
    }
```

0.3.6 Phase 6: Certificate Generation and Export (Months 7-9)

Objective: Generate machine-checkable certificates for all Bell inequalities and quantum bounds.

```
from dataclasses import dataclass, asdict
import json

@dataclass
class BellInequalityCertificate:
    """Certificate for a Bell inequality and its quantum violation."""

    # Scenario
    scenario: Tuple[int, int, int]  # (N, M, d)

    # Bell inequality
    inequality_name: str
    coefficients: List[str]  # Rational coefficients as strings "p/q"
    local_bound: str  # Rational

    # Quantum bound
    quantum_bound: float
    npa_level: int
    sdp_solver: str
    sdp_status: str

    # Optimal strategy
    optimal_state: List[complex]  # Quantum state |
    optimal_measurements: Dict  # {party: {x: [M_a for a in range(d)]}}
    achieved_value: float
    violation: float

    # Verification
    vertex_witness: List[int]  # Indices of deterministic strategies
        saturating local bound
    local_bound_verified: bool
    quantum_bound_verified: bool

    # Metadata
    timestamp: str
    computational_time: float

def generate_bell_certificate(scenario: BellScenario, bell_ineq: Dict,
                              npa_result: Dict, optimal_strategy: Dict)
                                 -> BellInequalityCertificate:
    """Generate complete certificate for Bell inequality."""

    import time
    from datetime import datetime

```

```python
    # Verify local bound by checking all vertices
    vertices = generate_deterministic_strategies(scenario)
    coeffs_float = [float(c) for c in bell_ineq['coefficients']]
    local_values = vertices @ np.array(coeffs_float)
    local_bound_computed = np.max(local_values)
    local_bound_expected = float(bell_ineq['bound'])
    local_verified = np.isclose(local_bound_computed,
        local_bound_expected, rtol=1e-6)

    # Vertices saturating the bound (within tolerance)
    vertex_witness = np.where(np.isclose(local_values,
        local_bound_computed, atol=1e-8))[0].tolist()

    # Verify quantum bound matches NPA result
    quantum_verified = np.isclose(optimal_strategy['bell_value'],
                                  npa_result['quantum_bound'],
                                  rtol=1e-4)

    cert = BellInequalityCertificate(
        scenario=(scenario.N, scenario.M, scenario.d),
        inequality_name="CHSH" if scenario.M == 2 and scenario.d == 2
            else "Custom",
        coefficients=[str(c) for c in bell_ineq['coefficients']],
        local_bound=str(bell_ineq['bound']),
        quantum_bound=npa_result['quantum_bound'],
        npa_level=npa_result['level'],
        sdp_solver="MOSEK",
        sdp_status=npa_result['status'],
        optimal_state=[complex(z) for z in optimal_strategy['state']],
        optimal_measurements=optimal_strategy['measurements'],
        achieved_value=optimal_strategy['bell_value'],
        violation=optimal_strategy['violation'],
        vertex_witness=vertex_witness,
        local_bound_verified=local_verified,
        quantum_bound_verified=quantum_verified,
        timestamp=datetime.now().isoformat(),
        computational_time=time.time()
    )

    return cert

def export_certificate_json(cert: BellInequalityCertificate, filepath:
    str):
    """Export certificate to JSON with exact arithmetic."""

    # Convert complex numbers to [real, imag] pairs
    def complex_to_list(z):
        return [z.real, z.imag]

    cert_dict = asdict(cert)
    cert_dict['optimal_state'] = [complex_to_list(z) for z in
        cert.optimal_state]

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

```

```python
    print(f"Certificate exported to {filepath}")

# Example: Full pipeline for CHSH
if __name__ == "__main__":
    scenario = BellScenario(N=2, M=2, d=2)

    # Step 1: Enumerate Bell inequalities
    bell_ineqs = enumerate_bell_inequalities(scenario)
    chsh_ineq = bell_ineqs[0]  # Assume first is CHSH (verify by inspection)

    # Step 2: Compute quantum bound via NPA
    npa_result = npa_sdp_chsh(level=2)

    # Step 3: Find optimal strategy
    optimal_strategy = chsh_quantum_optimal()  # Analytical for CHSH

    # Step 4: Generate certificate
    certificate = generate_bell_certificate(scenario, chsh_ineq, npa_result, optimal_strategy)

    # Step 5: Export
    export_certificate_json(certificate, "chsh_certificate.json")

    print(f"\nCertificate Summary:")
    print(f"  Local bound: {certificate.local_bound}")
    print(f"  Quantum bound: {certificate.quantum_bound:.8f}")
    print(f"  Violation: {certificate.violation:.8f}")
    print(f"  Verified: Local={certificate.local_bound_verified}, Quantum={certificate.quantum_bound_verified}")
```

0.4 4. Example Starting Prompt

Use this prompt to initialize a long-running AI system for Bell inequality research:

```
You are a theoretical physicist working on Bell inequalities and quantum nonlocality.
Your task is to systematically enumerate all Bell inequalities for small scenarios,
compute their quantum bounds via the NPA hierarchy, and certify device-independent
properties from observed violations.

CONTEXT:
Bell inequalities provide mathematical tests distinguishing quantum mechanics from
local hidden variable (LHV) theories. The CHSH inequality for two parties with binary
measurements states S = E( A B ) + E( A B ) + E( A B ) - E( A B ) 2 for all LHV,
while quantum mechanics achieves S = 2 2 (Tsirelson's bound). This quantum advantage
enables device-independent cryptography and randomness certification.

```

```
The NPA (Navarro-Pironio-Ac n) hierarchy systematically computes quantum bounds by
solving semidefinite programs (SDPs) that converge to the quantum set Q. Facet
enumeration of the local polytope L (via vertex enumeration) yields all possible
Bell inequalities for a given scenario.

OBJECTIVE:
Phase 1 (Months 1-2): Implement CHSH inequality, verify local bound = 2 and
  Tsirelson bound = 2 2  both analytically and numerically.

Phase 2 (Months 2-4): Implement NPA hierarchy at levels 1-3 using CVXPY + MOSEK.
  Verify convergence to Tsirelson bound for CHSH as level increases.

Phase 3 (Months 4-5): Enumerate all Bell inequalities for (2,2,2) scenario via
  vertex enumeration of local polytope. Identify CHSH among facets. Export all
  inequalities with rational coefficients.

Phase 4 (Months 5-6): For each Bell inequality, find optimal quantum strategy
  (state + measurements) achieving maximal violation. Extract from NPA moment matrix.

Phase 5 (Months 6-7): Implement device-independent randomness certification using
  Ac n et al. (2012) bounds for CHSH. Compute min-entropy H_min(A|E) from observed
  Bell violations.

Phase 6 (Months 7-9): Generate machine-checkable certificates for all results:
  - Vertex witnesses for local bounds (deterministic strategies saturating inequality)
  - SDP dual certificates for quantum bounds (NPA optimality)
  - Optimal quantum strategies with achieved violations
  - Export as JSON with exact rational arithmetic

PURE THOUGHT CONSTRAINTS:
- Use ONLY symbolic mathematics (sympy) for Bell inequality coefficients (exact rationals)
- All quantum bounds must come with SDP solver certificates (MOSEK dual optimum)
- Verify local bounds by exhaustive enumeration of deterministic strategies
- No experimental data until final benchmarking against published loophole-free tests
- All states and measurements must achieve violations within numerical tolerance    = 10

SUCCESS CRITERIA:
- Minimum Viable Result (2-4 months): CHSH working with verified
```

```
    bounds, NPA level 1-2
- Strong Result (6-8 months): All Bell inequalities for (2,2,2)
    enumerated, quantum
  bounds via NPA level 3, device-independent randomness certification
      operational
- Publication-Quality (9 months): Extension to (2,3,2) or (3,2,2)
    scenarios, novel
  Bell inequalities with optimal quantum strategies, comparison with
      experimental data
  from loophole-free tests

START:
Begin by implementing CHSH inequality verification (Phase 1). Write
    Python code using
numpy for correlations, verify local bound = 2 by checking all 16
    deterministic
strategies, then compute quantum optimal strategy using singlet state
    and measurements
at angles 0  , 45  , 90  , -45  . Export results with full numerical
    precision.
```

0.5 5. Success Criteria

0.5.1 Minimum Viable Result (MVR) - 2-4 Months

Core Functionality:

- **CHSH inequality implemented and verified: local bound = 2, Tsirelson = 22**
- **NPA hierarchy at level 1-2 operational with MOSEK solver**
- **Quantum optimal strategy for CHSH: singlet state + Pauli measurements**
- **Basic certificate generation with rational coefficients**

Deliverables:

- `chsh`$_v erification.py : Exhaustivecheckofall16deterministicstrategies$
- `npa`$_c hsh.py : NPASDPatlevels1-2, convergencetoTsirelsonboundwithin0.01$
- `chsh`$_c ertificate.json : Completecertificatewithstate, measurements, bounds$

Quality Metrics:

- **Local bound verified exactly: max over vertices = 2.0 (machine precision)**
- **Quantum bound within 10 of 22:** $|\mathrm{S}_N PA - 22| < 10$

- **SDP solver convergence in <10 seconds for level 2**

0.5.2 Strong Result - 6-8 Months

Extended Capabilities:

- **All Bell inequalities for (2,2,2) enumerated via convex hull (typically 8 facets up to symmetry)**
- **NPA hierarchy at level 3 achieving convergence within 10 for all inequalities**
- **Optimal quantum strategies extracted from moment matrices for all facets**
- **Device-independent randomness certification: H**$_m in(A|E) from CHSH violations$

- **Facets for (2,3,2) scenario (2 parties, 3 measurements, binary outcomes)**

Deliverables:

- `enumerate`$_f acets.py : Complete facet enumeration for (2,2,2) and (2,3,2)$
- `npa`$_g eneral.py : NPA at arbitrary level for arbitrary Bell inequality$
- `di`$_r andomness.py : Min - entropy bounds from observed correlations$
- `bell`$_d atabase.json : All Bell inequalities with quantum bounds and optimal strategies$

Quality Metrics:

- **All facets verified by d+1 affinely independent vertices saturating bound**
- **Quantum bounds converged: NPA level 3 within 10 of analytical Tsirelson bounds (where known)**
- **Device-independent randomness matches literature: CHSH=2.6 → H**$_m in 0.23 bits per round$

- **Computational time <5 minutes for (2,2,2) enumeration, <1 hour for (2,3,2)**

0.5.3 Publication-Quality Result - 9 Months

Novel Contributions:

- **Extension to (3,2,2) scenario (3 parties, Mermin-GHZ inequality)**
- **Tight quantum bounds for all facets of (2,3,2) and selected facets of (3,2,2)**
- **Device-independent entanglement certification: Schmidt rank bounds from violations**
- **Comparison with experimental data from loophole-free Bell tests (Hensen et al. 2015, Giustina et al. 2015)**
- **Novel Bell inequalities for asymmetric scenarios (e.g., (2,4,2) with non-trivial quantum advantage)**

Deliverables:

- `mermin`$_i nequality.py : GHZ state + Mermin inequality, quantum bound 4 vs local 2$
- `experimental`$_c omparison.py : Statistical analysis of published loophole - free test data$

- Research paper: "Complete Classification of Bell Inequalities for Small Scenarios via NPA Hierarchy"
- Interactive database: Web interface for querying Bell inequalities by scenario

Quality Metrics:

- All results for (2,3,2) match published Tsirelson bounds (Pironio et al., J. Math. Phys. 2005)
- Mermin inequality: quantum 4 vs local 2, achieved with GHZ state
- Experimental data analysis: statistical significance >5 for Bell violation in published datasets
- Novel results: At least 1 new Bell inequality with computed quantum bound not in literature
- Formal verification: Lean4 proofs for CHSH local bound and Tsirelson bound (optional advanced goal)

0.6 6. Verification Protocol

0.6.1 Automated Checks (Run After Every Phase)

```python
def verify_bell_certificate(cert: BellInequalityCertificate) -> Dict[str, bool]:
    """
    Comprehensive verification of Bell inequality certificate.

    Returns: Dictionary of Boolean checks (all must be True).
    """
    checks = {}
    scenario = BellScenario(*cert.scenario)

    # 1. Local bound verification
    vertices = generate_deterministic_strategies(scenario)
    coeffs = [float(sp.sympify(c)) for c in cert.coefficients]
    local_values = vertices @ np.array(coeffs)
    local_max = np.max(local_values)
    local_expected = float(sp.sympify(cert.local_bound))
    checks['local_bound_correct'] = np.isclose(local_max, local_expected, rtol=1e-6)

    # 2. Vertex witness verification
    for v_idx in cert.vertex_witness:
        v_value = vertices[v_idx] @ np.array(coeffs)
        checks[f'vertex_{v_idx}_saturates'] = np.isclose(v_value, local_expected, atol=1e-8)

    # 3. Quantum bound feasibility (achieved value    quantum bound)
```

```
    checks['quantum_feasible'] = cert.achieved_value <= cert.quantum_bound + 1e-6

    # 4. Optimal state normalization
    state = np.array(cert.optimal_state)
    checks['state_normalized'] = np.isclose(np.linalg.norm(state), 1.0, atol=1e-8)

    # 5. POVM constraints:  _a  M^x_a = I for each measurement x
    # (Requires reconstructing measurement operators from certificate)
    # Placeholder: assume measurements are valid if provided
    checks['measurements_valid'] = True

    # 6. Achieved value matches recomputation
    # P = compute_quantum_correlations(state, measurements, scenario)
    # recomputed_value = np.dot(coeffs, P.flatten())
    # checks['value_reproducible'] = np.isclose(recomputed_value, cert.achieved_value, rtol=1e-6)
    # (Requires full measurement reconstruction)
    checks['value_reproducible'] = True  # Placeholder

    # 7. Violation positivity (if quantum > local)
    expected_violation = cert.quantum_bound - local_expected
    checks['violation_positive'] = (expected_violation > 1e-6) == (cert.violation > 1e-6)

    return checks

# Example usage
certificate = generate_bell_certificate(...)  # From Phase 6
verification_results = verify_bell_certificate(certificate)
print("Verification Results:")
for check, passed in verification_results.items():
    status = "     PASS" if passed else "     FAIL"
    print(f"  {status}: {check}")

assert all(verification_results.values()), "Certificate verification failed!"
```

0.6.2 Cross-Validation Against Known Results

```
KNOWN_TSIRELSON_BOUNDS = {
    ('CHSH', (2,2,2)): 2 * np.sqrt(2),
    ('Mermin', (3,2,2)): 4.0,
    ('CGLMP_3', (2,2,3)): 2.9149,  # Approx. value from literature
}

def cross_validate_quantum_bounds(certificates: List[BellInequalityCertificate]):
    """Compare computed quantum bounds against published Tsirelson bounds."""
    for cert in certificates:
```

```
        key = (cert.inequality_name, cert.scenario)
        if key in KNOWN_TSIRELSON_BOUNDS:
            expected = KNOWN_TSIRELSON_BOUNDS[key]
            computed = cert.quantum_bound
            error = abs(computed - expected)
            print(f"{cert.inequality_name}: computed={computed:.8f}, expected={expected:.8f}, error={error:.2e}")
            assert error < 1e-5, f"Quantum bound mismatch for {cert.inequality_name}"
```

0.7 7. Resources and Milestones

0.7.1 Essential References

Foundational Papers:

- J.S. Bell, "On the Einstein-Podolsky-Rosen Paradox", Physics 1, 195 (1964)
- J.F. Clauser, M.A. Horne, A. Shimony, R.A. Holt, "Proposed Experiment to Test Local Hidden-Variable Theories", PRL 23, 880 (1969)
- B.S. Tsirelson, "Quantum Generalizations of Bell's Inequality", Lett. Math. Phys. 4, 93 (1980)

NPA Hierarchy:

- M. Navascués, S. Pironio, A. Acín, "A Convergent Hierarchy of Semidefinite Programs Characterizing the Set of Quantum Correlations", NJP 10, 073013 (2008)
- A.C. Doherty, Y.-C. Liang, B. Toner, S. Wehner, "The Quantum Moment Problem and Bounds on Entangled Multi-prover Games", Proc. IEEE CCC (2008)

Device-Independent Protocols:

- A. Acín et al., "Device-Independent Security of Quantum Cryptography against Collective Attacks", PRL 98, 230501 (2007)
- S. Pironio et al., "Random Numbers Certified by Bell's Theorem", Nature 464, 1021 (2010)

Reviews:

- N. Brunner, D. Cavalcanti, S. Pironio, V. Scarani, S. Wehner, "Bell Nonlocality", Rev. Mod. Phys. 86, 419 (2014) [Start here]
- R. Colbeck, R. Renner, "Free Randomness can be Amplified", Nature Phys. 8, 450 (2012)

Experimental Loophole-Free Tests:

- B. Hensen et al., "Loophole-free Bell Inequality Violation using Electron Spins Separated by 1.3 Kilometres", Nature 526, 682 (2015)
- M. Giustina et al., "Significant-Loophole-Free Test of Bell's Theorem with Entangled Photons", PRL 115, 250401 (2015)

0.7.2 Software Tools

- **CVXPY (v1.4+): Disciplined convex programming, interfaces with MOSEK**
- **MOSEK (v10+): Commercial SDP solver (free academic license), superior to open-source alternatives for Bell problems**
- **SymPy (v1.12+): Exact rational arithmetic for Bell inequality coefficients**
- **SciPy (spatial.ConvexHull): Facet enumeration via quickhull algorithm**
- **NumPy (v1.24+): Numerical linear algebra for moment matrices**

0.7.3 Common Pitfalls

- **Numerical Precision in SDP Solvers: MOSEK default tolerance (10) may be insufficient for tight bounds; increase via solver options**
- **Moment Matrix Ordering: Inconsistent indexing of operator sequences leads to wrong commutation constraints; use canonical ordering**
- **Vertex Enumeration Complexity: Exponential in scenario size; (3,3,2) has >10 vertices, infeasible for direct convex hull**
- **Optimal Strategy Extraction: Moment matrix gives expectations, not individual operators; reconstruction is non-unique and requires additional constraints**
- **Symmetry Exploitation: Bell inequalities form equivalence classes under local unitary rotations and permutations; reduce search space by symmetry**

0.7.4 Milestone Checklist

Month 2:

x **CHSH local bound = 2 verified by exhaustive enumeration**

x **CHSH quantum bound = 22 from NPA level 1-2**

x **Analytical optimal strategy (singlet + Pauli measurements)**

Month 4:

NPA level 3 operational for arbitrary Bell inequality

All 8 facets of (2,2,2) local polytope enumerated

Quantum bounds for all (2,2,2) facets converged to 10

Month 6:

Optimal quantum strategies extracted for all (2,2,2) facets

Device-independent randomness certification: $\mathbf{H}_m in(A|E) from CHSH$

Facet enumeration for (2,3,2) complete

Month 9:

Mermin inequality for (3,2,2): quantum 4 vs local 2

Experimental data analysis: CHSH violations in loophole-free tests

At least 1 novel Bell inequality with computed quantum bound

Research paper draft: "Complete Classification of Bell Inequalities for Small Scenarios"

End of PRD 22: Bell Inequalities and Quantum Nonlocality
Certificate-based pure thought investigation of quantum foundations, enabling device-independent cryptography and randomness expansion. All results verifiable via SDP duality and vertex enumeration.