```
# ------
# Hybrid Quantum Folding & Cyclization System (HQFCS) - v2
#
# This script saliently combines three physics-based chemical
synthesis simulations:
# 1. (NEW) Stage 0: Hydrocarbon Sequencing:
#    Analyzes the precursor's formula to determine chain length and
unsaturation.
# 2. Stage 1: Quantum-Informed Matter Folding (from dist_steroid.py):
#    Uses gamma backscatter and skyrmion spin dynamics to "fold" a
molecule.
# 3. Stage 2: Skyrmion-Optics Metamaterial Cyclization (from
Skyrmion-Optics...):
#    Uses circular dichroism and skyrmion-optics to "cyclize" a
precursor.
#
# Salient Combination (Full Pipeline S0 -> S1 -> S2):
# - STAGE 0: Sequences the precursor to find unsaturation/cyclization
sites.
# - (S0->S1 LINK): This sequence data *primes* the initial folding
parameters (curvature, entropy) for Stage 1.
# - STAGE 1: Folds the precursor based on the primed parameters.
# - (S1->S2 LINK): The final folded parameters *prime* the chemical
reaction state (catalyst, stability) for Stage 2.
# - (S1->S2 LINK): The final spin density from Stage 1 provides a
*spin boost* to Stage 2's tunneling efficiency.
# ------

import math
import random
import re # <-- Added for sequencing
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from typing import List, Tuple

# ------------------------------
# --- SHARED/HYBRID CLASSES ---
# ------------------------------

class Molecule:
    """Represents the hybrid target molecule for the entire
process."""
    def __init__(self, name, formula, target_ring_structure):
        self.name = name
        self.formula = formula
        self.target_ring_structure = target_ring_structure # From
Skyrmion-Optics target
```

```
# ------------------------------
# --- STAGE 0: SEQUENCING CLASSES (NEW) ---
# ------------------------------

class SequenceData:
    """Holds the results of the initial hydrocarbon sequence
analysis."""
    def __init__(self, carbon_chain_length: int, unsaturation_sites:
int, potential_cyclization_points: int):
        self.carbon_chain_length = carbon_chain_length
        self.unsaturation_sites = unsaturation_sites
        self.potential_cyclization_points =
potential_cyclization_points # Key new parameter

# ------------------------------
# --- STAGE 1: FOLDING CLASSES (from dist_steroid.py) ---
# ------------------------------

class GammaSignature:
    """Represents the gamma backscatter signature for folding."""
    def __init__(self, energy, angle):
        self.energy = energy
        self.angle = angle

class FoldingParams:
    """Internal parameters derived from the gamma scan."""
    def __init__(self, phase_shift, curvature, entropy_bias):
        self.phase_shift = phase_shift
        self.curvature = curvature
        self.entropy_bias = entropy_bias

class LaserConfig:
    """Configuration for the matter-folding laser."""
    def __init__(self, wavelength, pulse_width, coherence):
        self.wavelength = wavelength
        self.pulse_width = pulse_width
        self.coherence = coherence

# ------------------------------
# --- STAGE 2: CYCLIZATION CLASSES (from Skyrmion-Optics) ---
# ------------------------------

class SpectrumSignature:
    """Represents the Circular Dichroism signature of the current
conformation."""
    def __init__(self, ellipticity: float, absorption_peak: float):
        self.ellipticity = ellipticity     # Handedness/chirality
```

```
signal (mdeg)
        self.absorption_peak = absorption_peak # Wavelength (nm)

class MetamaterialParams:
    """Parameters defining the graphene-based stabilization
scaffold."""
    def __init__(self, tensile_stress: float, pore_density: float,
confinement_rigidity: float):
        self.tensile_stress = tensile_stress
        self.pore_density = pore_density
        self.confinement_rigidity = confinement_rigidity # How much
the scaffold resists structural change

class ReactionState:
    """Parameters defining the chemical environment and molecular
preparation."""
    def __init__(self, catalyst_activity: float,
intermediate_stability: float):
        self.catalyst_activity = catalyst_activity  # E.g., Lewis acid
strength (units)
        self.intermediate_stability = intermediate_stability # E.g.,
Solvent-mediated stabilization (units)

class SkyrmionGateConfig:
    """Parameters for the Tunable Skyrmion Field and the Chiral
Optical Gate."""
    def __init__(self, magnetic_field: float, skyrmion_density: float,
optical_polarization: float):
        self.magnetic_field = magnetic_field         # Controls
Skyrmion size/stability (Tesla)
        self.skyrmion_density = skyrmion_density     # Density of the
magnetic 'quanta' (nm^-2)
        self.optical_polarization = optical_polarization # Chiral
light input (-1.0 to 1.0)

# ------------------------------
# Initialization
# ------------------------------

def initialize_system():
    print("[INIT] Hybrid Quantum Folding & Cyclization System (HQFCS)
v2 initialized.\n")

# ------------------------------------------------------------------
# --- STAGE 0: SEQUENCING FUNCTIONS (NEW) ---
# ------------------------------------------------------------------

def sequence_hydrocarbon_precursor(molecule: Molecule) ->
```

```
SequenceData:
    """
    Simulates a basic analysis of the precursor's hydrocarbon
skeleton.
    This is the new 'Stage 0' analysis.
    """
    print(f"\n[STAGE 0] Sequencing precursor: {molecule.name}
({molecule.formula})")
    try:
        # This is a simplified simulation. A real one would be vastly
complex.
        c_search = re.search(r'C(\d+)', molecule.formula)
        h_search = re.search(r'H(\d+)', molecule.formula)

        c_count = int(c_search.group(1)) if c_search else 0
        h_count = int(h_search.group(1)) if h_search else 0

        if c_count == 0 or h_count == 0:
            raise ValueError("Incomplete formula for sequencing.")

        # Calculate Degree of Unsaturation (DoU)
        # DoU = C + 1 - (H/2) - (X/2) + (N/2)
        # Assuming only C and H for hydrocarbons:
        max_h = 2 * c_count + 2
        unsaturation_sites = (max_h - h_count) // 2

        # Simulate potential cyclization points based on chain length
and unsaturation
        # More sites = more places for the folding to target.
        potential_cyclization_points = max(1, unsaturation_sites // 2)
+ (c_count // 10)

        print(f"[SEQ-S0] Carbon Chain: {c_count}")
        print(f"[SEQ-S0] Unsaturation Sites (DoU):
{unsaturation_sites}")
        print(f"[SEQ-S0] Potential Cyclization Points:
{potential_cyclization_points}")

        return SequenceData(c_count, unsaturation_sites,
potential_cyclization_points)

    except Exception as e:
        print(f"[SEQ-S0-ERROR] Could not parse formula
{molecule.formula}: {e}")
        print("[SEQ-S0-ERROR] Using default sequence values.")
        # Return default values
        return SequenceData(20, 4, 2)
```

```
# ------------------------------------------------------------
# --- STAGE 1: FOLDING FUNCTIONS (from dist_steroid.py) ---
# ------------------------------------------------------------

def capture_gamma_backscatter(molecule: Molecule) -> GammaSignature:
    """Simulates gamma scan for initial folding."""
    print(f"[SCAN-S1] Scanning molecule: {molecule.name}
({molecule.formula}) for folding.")
    energy = round(random.uniform(1.0, 1.5), 2)
    angle = round(random.uniform(30.0, 60.0), 2)
    return GammaSignature(energy, angle)

def interpret_folding_params(sig: GammaSignature, seq_data:
SequenceData) -> FoldingParams:
    """
    Interprets gamma signature to get folding parameters.
    *** SALIENT COMBINATION (S0 -> S1) ***
    Uses SequenceData from Stage 0 to "prime" the initial folding
entropy and curvature.
    """
    base_phase_shift = sig.energy * 0.85
    base_curvature = math.tan(math.radians(sig.angle)) * 0.1
    base_entropy_bias = 1.0 / (1.0 + sig.energy)

    # --- SALIENT COMBINATION (S0->S1) ---
    # More unsaturation sites (from Stage 0) = more flexibility =
higher initial entropy to control.
    entropy_boost = seq_data.unsaturation_sites * 0.05
    # More cyclization points (from Stage 0) = needs a more complex
fold = higher target curvature.
    curvature_boost = seq_data.potential_cyclization_points * 0.02

    final_entropy_bias = base_entropy_bias + entropy_boost
    final_curvature = base_curvature + curvature_boost

    print(f"[PRIME-S1] Stage 0 Params (Unsaturation:
{seq_data.unsaturation_sites}, Cycl. Points:
{seq_data.potential_cyclization_points})")
    print(f"[PRIME-S1] Primed Folding State (Curvature:
{round(final_curvature, 3)}, Entropy: {round(final_entropy_bias,
3)})")
    # --- END COMBINATION ---

    return FoldingParams(base_phase_shift, final_curvature,
final_entropy_bias)

def derive_laser_geometry(params: FoldingParams) -> LaserConfig:
    """Configures laser based on folding parameters."""
```

```
    wavelength = 400.0 + params.phase_shift * 10.0
    pulse_width = 50.0 - params.curvature * 5.0
    coherence = 1.0 - params.entropy_bias
    return LaserConfig(wavelength, pulse_width, coherence)

def recalculate_band_states(params: FoldingParams, config:
LaserConfig) -> Tuple[float, float]:
    """Recalculates electron band states during folding."""
    print("[BAND-S1] Recalculating electron band states...")
    band_gap = max(1.5 - (params.curvature * 2.0 + params.entropy_bias
* 1.2), 0.5)
    orbital_shift = config.coherence * 0.3
    print(f"[BAND-S1] Band Gap: {round(band_gap, 3)} eV | Orbital
Shift: {round(orbital_shift, 3)} units")
    return band_gap, orbital_shift

def simulate_skyrmion_dynamics(iteration: int) -> float:
    """Simulates skyrmion tunneling dynamics for folding stability."""
    print(f"[SKYRMION-S1] Simulating tunneling dynamics at iteration
{iteration}...")
    position = np.sin(iteration * 0.5) * 5
    tunneling_current = 0.8 + np.cos(iteration * 0.3) * 0.2
    spin_density = np.exp(-abs(position)) * tunneling_current
    print(f"[SKYRMION-S1] Position: {round(position,2)} | Current:
{round(tunneling_current,2)} | Spin Density: {round(spin_density,3)}")
    return spin_density

def fold_matter(config: LaserConfig) -> bool:
    """Executes the laser-based matter folding."""
    if config.coherence < 0.5:
        print("[ERROR-S1] Laser coherence too low. Folding
aborted.\n")
        return False
    print("[SUCCESS-S1] Folding completed. Molecular bonds
realigned.\n")
    return True

def run_folding_stage(molecule: Molecule, seq_data: SequenceData) ->
Tuple[LaserConfig, FoldingParams, List[float], List[float],
List[float], bool]:
    """
    The adaptive loop for Stage 1: Folding.
    Now requires seq_data to prime the parameters.
    """
    sig = capture_gamma_backscatter(molecule)
    # Pass seq_data to the interpreter
    params = interpret_folding_params(sig, seq_data) # (modified)
    config = derive_laser_geometry(params)
```

```
    band_gap_history = []
    orbital_shift_history = []
    spin_density_history = []
    success_flag = False

    for i in range(3):
        print(f"[LOOP-S1] Iteration {i+1}: Evaluating system
stability...")

        spin_density = simulate_skyrmion_dynamics(i)
        spin_density_history.append(spin_density)

        band_gap, orbital_shift = recalculate_band_states(params,
config)
        band_gap_history.append(band_gap)
        orbital_shift_history.append(orbital_shift)

        if spin_density < 0.7:
            print("[ADAPT-S1] Spin density low. Enhancing curvature.")
            params.curvature *= 1.1
            config = derive_laser_geometry(params)

        if config.coherence < 0.6 or band_gap < 0.7:
            print("[ADAPT-S1] Coherence or band gap unstable.
Adjusting entropy bias.")
            params.entropy_bias *= 0.9
            config = derive_laser_geometry(params)

        if fold_matter(config):
            print(f"[RESULT-S1] {molecule.name} successfully folded
with skyrmion-guided control.\n")
            success_flag = True
            return config, params, band_gap_history,
orbital_shift_history, spin_density_history, success_flag

        print("[RETRY-S1] Folding failed. Retrying...\n")

    print(f"[RESULT-S1] {molecule.name} folding stage complete
(unstable).\n")
    return config, params, band_gap_history, orbital_shift_history,
spin_density_history, success_flag

# ----------------------------------------------------------------------
# --- STAGE 2: CYCLIZATION FUNCTIONS (from Skyrmion-Optics) ---
# ----------------------------------------------------------------------

def capture_cd_spectrum(target: Molecule) -> SpectrumSignature:
```

```
    """Simulates capturing the CD spectrum to determine chiral state
of folded precursor."""
    print(f"[CD-SCAN-S2] Scanning folded precursor: {target.name} for
{target.target_ring_structure} cyclization.")
    ellipticity = round(random.uniform(-15.0, 15.0), 1)
    peak = round(random.uniform(280.0, 320.0), 1)
    return SpectrumSignature(ellipticity, peak)

def interpret_cyclization_params(sig: SpectrumSignature, fold_params:
FoldingParams) -> Tuple[MetamaterialParams, ReactionState]:
    """
    Interprets the signature to derive metamaterial and reaction
parameters.
    *** SALIENT COMBINATION (S1 -> S2) ***
    Uses FoldingParams from Stage 1 to "prime" the initial
ReactionState.
    """

    # Metamaterial Derivation (Physical confinement)
    rigidity = 1.0 + abs(sig.ellipticity) / 10.0
    tensile_stress = sig.absorption_peak / 300.0 * 1.2
    pore_density = 0.5 + sig.absorption_peak * sig.ellipticity *
0.0001
    meta_params = MetamaterialParams(tensile_stress, pore_density,
rigidity)

    # Initial Reaction State Derivation (Chemically primed)
    # Base catalyst activity from CD scan
    base_catalyst = max(1.5 - abs(sig.ellipticity) * 0.05, 0.5)
    # Base stability from CD scan
    base_stability = sig.absorption_peak / 300.0 * 1.5

    # --- SALIENT COMBINATION (S1->S2) ---
    # Folding (high curvature, low entropy) enhances chemical state
    # High curvature from S1 implies a well-folded, stable
intermediate
    curvature_boost = fold_params.curvature * 0.5
    # Low entropy bias from S1 implies better alignment for catalysis
    entropy_dampen = fold_params.entropy_bias * 0.2

    catalyst_activity = max(base_catalyst - entropy_dampen, 0.3)
    intermediate_stability = base_stability + curvature_boost
    print(f"[PRIME-S2] Stage 1 Params (Curvature:
{round(fold_params.curvature,2)}, Entropy:
{round(fold_params.entropy_bias,2)})")
    print(f"[PRIME-S2] Primed Chemical State (Activity:
{round(catalyst_activity,2)}, Stability:
{round(intermediate_stability,2)})")
```

```
    # --- END COMBINATION ---

    reaction_state = ReactionState(catalyst_activity,
intermediate_stability)

    return meta_params, reaction_state

def derive_skyrmion_gate_geometry(params: MetamaterialParams, sig:
SpectrumSignature) -> SkyrmionGateConfig:
    """Derives magnetic field and optical gate parameters."""
    magnetic_field = 0.8 + params.confinement_rigidity * 0.3
    skyrmion_density = math.log(params.tensile_stress + 1.0) * 5.0
    optical_polarization = -sig.ellipticity / 15.0

    return SkyrmionGateConfig(magnetic_field, skyrmion_density,
optical_polarization)

def recalculate_tunneling_states(meta_params: MetamaterialParams,
skyrmion_config: SkyrmionGateConfig, reaction_state: ReactionState) ->
Tuple[float, float, float]:
    """
    Determines optical mode coupling efficiency and transport loss,
    introducing CHEMICAL INFLUENCE on coupling.
    """
    print("[RECAL-S2] Recalculating Skyrmion-Photonic and Chemical
states...")

    # Base Coupling Efficiency (Magnetic/Optical)
    base_coupling = min(skyrmion_config.skyrmion_density / 8.0, 0.98)
* (1.0 - abs(skyrmion_config.optical_polarization) * 0.1)

    # Chemical Influence Factor
    chemical_influence = 1.0 + (reaction_state.catalyst_activity *
reaction_state.intermediate_stability) * 0.1

    # Combined Coupling Efficiency
    combined_coupling = min(base_coupling * chemical_influence, 0.99)

    # Transport loss (Physical)
    transport_loss = max(0.01 + 1.0 /
(meta_params.confinement_rigidity * 10.0), 0.05)

    print(f"[RECAL-S2] Chemical Influence Factor:
{round(chemical_influence, 3)}")
    print(f"[RECAL-S2] Combined Coupling Efficiency:
{round(combined_coupling, 3)}")
    return combined_coupling, transport_loss, chemical_influence
```

```
def simulate_tunneling_dynamics(iteration: int, chemical_influence:
float, final_fold_spin_density: float) -> float:
    """
    Simulates the final efficiency of the light transport.
    *** SALIENT COMBINATION (S1 -> S2) ***
    Uses final_fold_spin_density from Stage 1 to provide a "spin
boost".
    """
    print(f"[SKYOP-S2] Simulating Skyrmion-Optics tunneling efficiency
at iteration {iteration}...")

    skyrmion_stability = 0.8 + np.sin(iteration * 0.4) * 0.15

    # --- SALIENT COMBINATION (S1->S2) ---
    # Spin density from Stage 1 provides a base-level boost to
tunneling
    spin_boost = final_fold_spin_density * 0.1

    # Total effective efficiency (boosted by chemistry AND prior spin
state)
    tunneling_efficiency = max(0.9 * skyrmion_stability *
chemical_influence + spin_boost - iteration * 0.05, 0.5) # (modified)

    print(f"[SKYOP-S2] (Fold Spin Boost: {round(spin_boost, 3)}) |
Tunneling Efficiency: {round(tunneling_efficiency, 3)}") # (modified)
    return tunneling_efficiency

def execute_cyclization(config: SkyrmionGateConfig, efficiency: float)
-> Tuple[bool, float, float]:
    """Attempts the final cyclization, incorporating tunneling
efficiency."""

    cyclization_yield = efficiency * random.uniform(0.85, 0.95)
    enantiomeric_excess = abs(config.optical_polarization) *
random.uniform(0.90, 0.99) * 100.0

    if efficiency < 0.75:
        print("[ERROR-S2] Optical Tunneling Efficiency too low.
Cyclization aborted.\n")
        return False, 0.0, 0.0

    print(f"[SUCCESS-S2] Photochemical cyclization completed via
Skyrmion-guided light.")
    return True, cyclization_yield, enantiomeric_excess

def run_cyclization_stage(target: Molecule, fold_params:
FoldingParams, final_fold_spin_density: float) ->
Tuple[SkyrmionGateConfig, MetamaterialParams, ReactionState,
```

```
List[float], List[float], List[float], float, float, List[float]]:
    """The iterative loop for Stage 2: Cyclization."""

    sig = capture_cd_spectrum(target)
    # Pass fold_params from S1 into the S2 interpreter
    meta_params, reaction_state = interpret_cyclization_params(sig,
fold_params) # (modified)
    skyrmion_config = derive_skyrmion_gate_geometry(meta_params, sig)
#

    coupling_history = []
    loss_history = []
    efficiency_history = []
    chemical_influence_history = [] #

    final_yield, final_ee = 0.0, 0.0

    for i in range(5): #
        print(f"[LOOP-S2] Iteration {i+1}: Evaluating Skyrmion-Optics
Transport...")

        combined_coupling, transport_loss, chemical_influence =
recalculate_tunneling_states(meta_params, skyrmion_config,
reaction_state) #
        coupling_history.append(combined_coupling)
        loss_history.append(transport_loss)
        chemical_influence_history.append(chemical_influence)

        # Pass final_fold_spin_density from S1 into S2 simulation
        tunneling_efficiency = simulate_tunneling_dynamics(i,
chemical_influence, final_fold_spin_density) # (modified)
        efficiency_history.append(tunneling_efficiency)

        # --- Adaptive Tuning based on Efficiency Feedback ---

        if tunneling_efficiency < 0.85:
            print("[ADAPT-S2] Tunneling efficiency low. Prioritizing
chemical optimization.")
            # TUNE 1: Chemical
            reaction_state.intermediate_stability *= 1.08
            reaction_state.intermediate_stability =
np.clip(reaction_state.intermediate_stability, 0.1, 2.5)
            # TUNE 2: Magnetic
            skyrmion_config.skyrmion_density *= 1.05

        elif combined_coupling < 0.8:
            print("[ADAPT-S2] Low coupling efficiency. Adjusting
catalyst for alignment.")
```

```
            # TUNE 3: Chemical
            reaction_state.catalyst_activity += random.uniform(-0.05,
0.05)
            reaction_state.catalyst_activity =
np.clip(reaction_state.catalyst_activity, 0.5, 2.0)
            # TUNE 4: Physical
            meta_params.confinement_rigidity *= 1.01
            skyrmion_config =
derive_skyrmion_gate_geometry(meta_params, sig) #

        success, current_yield, current_ee =
execute_cyclization(skyrmion_config, tunneling_efficiency) #

        if success and current_ee > 95.0 and current_yield > 0.90: #
            final_yield = current_yield
            final_ee = current_ee
            print(f"[RESULT-S2] {target.name} successfully cyclized to
optimal parameters. Yield: {round(final_yield*100, 1)}%, EE:
{round(final_ee, 1)}%.\n")
            return (skyrmion_config, meta_params, reaction_state,
coupling_history,
                    loss_history, efficiency_history, final_yield,
final_ee, chemical_influence_history)

        print("[RETRY-S2] Cyclization not optimal. Retrying...\n")

    # Return final best attempt
    return (skyrmion_config, meta_params, reaction_state,
coupling_history,
            loss_history, efficiency_history, final_yield, final_ee,
chemical_influence_history)

# ------------------------------
# Visualization Routines
# ------------------------------

def visualize_folding_skyrmions(molecule: Molecule,
spin_density_history: List[float]):
    """Visualizes the 3D evolution of Skyrmion dynamics from Stage
1."""
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    iterations = list(range(1, len(spin_density_history) + 1))
    # Note: Using S1's simulation params for position/current
    positions = [np.sin(i * 0.5) * 5 for i in iterations]
    currents = [0.8 + np.cos(i * 0.3) * 0.2 for i in iterations]

    ax.plot(positions, currents, spin_density_history, marker='o',
```

```
color='teal')
    ax.set_title(f"Stage 1 Skyrmion Evolution: {molecule.name}")
    ax.set_xlabel("Position")
    ax.set_ylabel("Tunneling Current")
    ax.set_zlabel("Spin Density")

    plt.tight_layout()
    plt.show()

def visualize_folding_stages(molecule: Molecule):
    """ASCII visualization of the Stage 1: Folding pipeline."""
    print(f"\n[ASCII] Stage 1: Folding Stages for {molecule.name}")
    print("""
    +-----------------------------+
    |   [Sterol Precursor Input]  |
    +-----------------------------+
                |
                v
    +-----------------------------+
    | [Gamma Backscatter Scan]    |
    +-----------------------------+
                |
                v
    +-----------------------------+
    | [Laser Geometry Tuning]     |
    +-----------------------------+
                |
                v
    +-----------------------------+
    | [Skyrmion Tunneling (S1)]   |
    +-----------------------------+
                |
                v
    +-----------------------------+
    | [Matter Folding Execution] |
    +-----------------------------+
                |
                v
    +-----------------------------+
    |  [Folded Precursor Output]  |
    +-----------------------------+
    """)

def visualize_cyclization_tunneling(target: Molecule,
efficiency_history: List[float], chemical_influence_history:
List[float]):
    """Visualizes the Skyrmion-Optics Tunneling Efficiency from Stage
2."""
```

```
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    iterations = list(range(1, len(efficiency_history) + 1))

    # Approximate skyrmion density growth over iterations
    skyrmion_density = [4.0 + np.sin(i*0.6) * 1.5 + i*0.5 for i in
iterations]

    ax.plot(skyrmion_density[:len(chemical_influence_history)],
chemical_influence_history, efficiency_history, marker='o',
color='lightgreen')
    ax.set_title(f"Stage 2 Transport Rate vs. Chem/Mag Control:
{target.name}")
    ax.set_xlabel("Skyrmion Density (nm^-2)")
    ax.set_ylabel("Chemical Influence Factor")
    ax.set_zlabel("Tunneling Efficiency")

    plt.tight_layout()
    plt.show()

def visualize_cyclization_stages(target: Molecule):
    """ASCII visualization of the Stage 2: Cyclization pipeline."""
    print(f"\n[ASCII] Stage 2: Chemically-Enhanced Cyclization Stages
for {target.name}")
    print("""
    +-----------------------------+
    | [Folded Precursor Input]    |
[source: 23] +-----------------------------+
                |
[source: 24] v
    +-----------------------------+
    | [CD Spectrum (Conformation)]|
[source: 25] +-----------------------------+
                |
[source: 26] v
    +-----------------------------+
    | [Metamaterial & Chemical Setup]|
[source: 27] +-----------------------------+
                |
[source: 28] v
    +-----------------------------+
    | [Skyrmion Field & Gate Setup]|
[source: 29] +-----------------------------+
                |
[source: 30] v
    +-----------------------------+
    | [TUNING: Chemical & Photonic]|
```

```
[source: 31] +-----------------------------+
                |
[source: 32] v
    +-----------------------------+
    | [Photochemical Cyclization] |
[source: 33] +-----------------------------+
                |
[source: 34] v
    +-----------------------------+
    | [Cyclized Chiral Product]   |
[source: 35] +-----------------------------+
    """)

# -------------------------------
# Combined Commentary
# -------------------------------

def master_commentary(molecule: Molecule, seq_data: SequenceData,
fold_results, cycle_results):
    """
    Provides a unified summary of the full hybrid synthesis run.
    Now includes Stage 0 data.
    """

    # Unpack results
    (fold_config, fold_params, band_gap_hist, orbital_hist, spin_hist,
fold_success) = fold_results
    (cycle_config, meta_params, react_state, coupling_hist, loss_hist,
eff_hist, final_yield, final_ee, _) = cycle_results

    final_fold_spin = spin_hist[-1] if spin_hist else 0.0
    final_fold_gap = band_gap_hist[-1] if band_gap_hist else 0.0

    final_cycle_eff = eff_hist[-1] if eff_hist else 0.0


print(f"\n\n============================================================")
    print(f"HYBRID SYNTHESIS REPORT: {molecule.name}")

print(f"============================================================\n")

    print("--- STAGE 0: PRECURSOR SEQUENCING ---")
    print(f"[STATUS] ANALYSIS COMPLETE.")
    print(f"    Carbon Chain Length: {seq_data.carbon_chain_length}")
    print(f"    Unsaturation Sites (DoU):
{seq_data.unsaturation_sites}")
    print(f"    Potential Cyclization Points:
{seq_data.potential_cyclization_points}")
```

```
    print("\n--- STAGE 1: MATTER FOLDING (Steroid Pre-alignment) ---")
    if not fold_success:
        print("[STATUS] FOLDING FAILED. Laser coherence was
unstable.")
        print("         Cyclization (Stage 2) proceeded with a
sub-optimal precursor conformation.")
    else:
        print("[STATUS] FOLDING SUCCESSFUL.")

    print(f"    Final Laser Coherence: {round(fold_config.coherence,
3)}")
    print(f"    Final Band Gap: {round(final_fold_gap, 3)} eV")
    print(f"    Final S1 Spin Density: {round(final_fold_spin, 3)}")

    print("\n--- STAGE 2: SKYRMION-OPTICS CYCLIZATION ---")
    if final_yield > 0:
        print(f"[STATUS] CYCLIZATION SUCCESSFUL.")
        print(f"    Final Yield: {round(final_yield * 100, 1)}%") #
(adapted)
        print(f"    Final EE: {round(final_ee, 1)}%") # (adapted)
    else:
        print("[STATUS] CYCLIZATION FAILED. Tunneling efficiency
remained too low.")

    print(f"    Final Catalyst Activity:
{round(react_state.catalyst_activity, 3)} units") #
    print(f"    Final Intermediate Stability:
{round(react_state.intermediate_stability, 3)} units") #
    print(f"    Final S2 Tunneling Efficiency: {round(final_cycle_eff,
3)}") #

    print("\n--- SALIENT ANALYSIS (Full Chain S0 -> S1 -> S2) ---")
    print(f"""
    The synthesis began with Stage 0 (Sequencing), which identified
    {seq_data.unsaturation_sites} unsaturation sites and
{seq_data.potential_cyclization_points} potential cyclization points.

    This sequence data was used to 'prime' Stage 1 (Folding),
    influencing its initial Curvature and Entropy Bias.

    The final folded parameters from Stage 1 (Curvature:
{round(fold_params.curvature, 2)},
    Entropy Bias: {round(fold_params.entropy_bias, 2)}) were then used
to 'prime'
    the chemical environment for Stage 2 (Cyclization), enhancing
Intermediate Stability.
```

```
    Finally, the Stage 1 spin density ({round(final_fold_spin, 3)})
provided a quantum
    'spin boost' to Stage 2's optical tunneling.

    This demonstrates a full-chain, multi-stage salient combination,
    where the output of each stage directly informs the initial
    conditions of the next.
    """)

# ------------------------------
# Main Execution
# ------------------------------

def main():
    initialize_system()

    molecules = [
        Molecule("Sterol Precursor-A", "C27H44", "Diels-Alder Ring"),
# (adapted) DoU = (2*27+2 - 44)/2 = 6
        Molecule("Ergosterol Precursor", "C28H42", "Macro-Lactam
Ring"), # (adapted) DoU = (2*28+2 - 42)/2 = 8
        Molecule("Prosta-Precursor-X", "C20H30O2", "Five-Membered
Ring") # (adapted) DoU = (2*20+2 - 30)/2 = 6 (Ignoring O)
    ]

    for mol in molecules:
        print(f"\n--- Processing Hybrid Target: {mol.name}
({mol.formula}) ---")

        # --- STAGE 0 (NEW) ---
        seq_data = sequence_hydrocarbon_precursor(mol)

        # --- STAGE 1 ---
        print("\n[STAGE 1] Initiating Quantum-Informed Matter
Folding...")
        # Pass seq_data into Stage 1
        fold_results = run_folding_stage(mol, seq_data) # (modified)
        (fold_config, fold_params, _, _, spin_history, fold_success) =
fold_results

        # Visualize Stage 1
        visualize_folding_stages(mol)
        if spin_history:
            visualize_folding_skyrmions(mol, spin_history)

        # --- STAGE 2 ---
        print(f"\n[STAGE 2] Initiating Skyrmion-Optics Cyclization
(Target: {mol.target_ring_structure})...")
```

```
        # Get the *last* spin density from stage 1 to feed into stage
2
        final_spin_density = spin_history[-1] if spin_history else 0.0

        # Pass folding params and spin density into stage 2
        cycle_results = run_cyclization_stage(mol, fold_params,
final_spin_density) # (adapted)

        # Unpack results needed for visualization and commentary
        (cycle_config, meta_params, react_state, coupling_hist,
loss_hist,
         efficiency_history, final_yield, final_ee,
chem_influence_history) = cycle_results

        # Visualize Stage 2
        visualize_cyclization_stages(mol) #
        if efficiency_history and chem_influence_history:
            visualize_cyclization_tunneling(mol, efficiency_history,
chem_influence_history) #

        # --- REPORT ---
        master_commentary(mol, seq_data, fold_results, cycle_results)
# (adapted)

    print("\n[END] Hybrid synthesis pipeline complete.")

if __name__ == "__main__":
    main()
```