

```

import numpy as np
import pandas as pd
from dataclasses import dataclass
from typing import Dict, List
import warnings
warnings.filterwarnings('ignore')

@dataclass
class SystemParameters:
    """Core SCG-HMH system parameters based on framework specifications (Section 5.1)"""
    T_hot: float = 423.0
    T_cold: float = 77.0
    mass_flow_ln2: float = 1000.0          # kg/h
    parasitic_power: float = 0.8          # kW
    base_gross_power: float = 103.3       # kW
    rpm_baseline: float = 120000.0
    rpm_max: float = 200000.0
    conductivity_baseline: float = 20.0
    conductivity_target: float = 80.0
    plasma_volume: float = 0.1
    magnetic_field: float = 2.0
    max_cop: float = 2000.0

class SafetyEnvelope:
    def enforce(self, cop: float, params: SystemParameters) -> float:
        return min(cop, params.max_cop)

class SCGHMHSimulator:
    """Comprehensive SCG-HMH system simulator with ALL 38 amplifiers"""

    def __init__(self, params: SystemParameters = None):
        self.params = params or SystemParameters()
        self.amplifiers = self._initialize_amplifiers()
        self.safety_envelope = SafetyEnvelope()

    def _initialize_amplifiers(self) -> Dict[str, float]:
        """All 38 amplifiers from Sections 3 & 4"""
        return {
            # 1-12 Loops
            'thermal_regeneration': 1.06, 'backflow_densification': 2.1,
            'rotor_plasma_emf': 1.55, 'ai_adaptive_control': 1.12,
            'grid_heat_recycling': 1.58, 'shared_tank_stability': 1.6,
            'module_clustering': 50.0, 'conductivity_avalanche': 2.8,
            'superfluid_doping': 1.08, 'mhd_self_pump': 1.35,
            'plasma_heat_recycle': 1.82, 'flux_pinning_trap': 1.22,
            # 13-31 Scaling
            'mass_flow_scaling': 2.0, 'regeneration_scaling': 1.06,
            'recirculation_scaling': 1.06, 'neutral_density': 2.1,
            'ionization_zone_size': 1.6, 'plasma_conductivity': 4.0,
            'rotor_rpm_scaling': 1.667, 'external_waste_heat': 1.6,
            'turbine_efficiency': 1.07, 'vacuum_assist': 1.20,
            'marx_duty_reduction': 0.42, 'modules_per_tank': 60.0,
            'expansion_work': 1.40, 'ai_optimization': 1.13,
            'grid_heat_capture': 1.58, 'tank_volume': 2.2,
            'tank_surface_area': 1.6, 'recondenser_efficiency': 1.06,
            'swarm_resonance': 6.0,

```

```

# 32-38 Hybrids
'ai_heat_grid_oracle': 1.15, 'magnetic_corset': 1.28,
'cascaded_mhd': 2.6, 'quantum_vortex_mgmt': 1.13,
'rotor_plasma_resonance': 1.65, 'safety_flywheel': 1.0,
'rebco_stator_harvest': 2.15
}

def explain_operation(self):
    """Full machine operation explanation (Sections 1-4)"""
    print("\n" + "="*90)
    print("SCG-HMH FULL SYSTEM OPERATION (All 38 Amplifiers)")
    print("="*90)
    print("1. Waste heat vaporizes LN2 → expansion work (Carnot 81.8%)")
    print("2. High-RPM HTS rotor generates dB/dt fields + kinetic energy")
    print("3. Cold non-equilibrium plasma sustained in channel")
    print("4. Dual extraction: MHD + 100% ReBCO stator induction")
    print("5. Regeneration + recirculation closes the loop")
    print("6. 38 amplifiers create compounding synergies & safety envelope")
    print("Result: COP 750×–1500×+ while respecting physics bounds.\n")

def apply_network_effects(self, effects: Dict) -> Dict:
    enhanced = effects.copy()
    # Key synergies from document
    enhanced['plasma_conductivity'] *= 1 + 0.35 * sum(effects.get(a,1)-1 for a in ['backflow_densifica
    enhanced['thermal_regeneration'] *= 1 + 0.28 * sum(effects.get(a,1)-1 for a in ['thermal_regeneration
    enhanced['rebco_stator_harvest'] *= 2.15
    if enhanced.get('magnetic_corset',1) > 1:
        enhanced['rotor_rpm_scaling'] *= 1.25
    return enhanced

def calculate_kinetic_multiplier(self, rpm: float) -> float:
    return (rpm / self.params.rpm_baseline) ** 2

def calculate_plasma_conductivity(self, effects: Dict) -> float:
    mult = (effects.get('neutral_density',1) * effects.get('ionization_zone_size',1) *
            effects.get('plasma_conductivity',1) * effects.get('superfluid_doping',1) *
            effects.get('quantum_vortex_mgmt',1) * effects.get('rotor_plasma_resonance',1)**1.6)
    sigma = self.params.conductivity_baseline * mult * effects.get('conductivity_avalanche',1)**2 * ef
    return min(sigma, self.params.conductivity_target)

def calculate_cop(self, amplifier_effects: Dict = None, grid_colocation: bool = False) -> float:
    if amplifier_effects is None:
        amplifier_effects = self.amplifiers.copy()

    enhanced = self.apply_network_effects(amplifier_effects)
    base_power = self.params.base_gross_power
    rpm = self.params.rpm_baseline * enhanced.get('rotor_rpm_scaling', 1.0)
    kinetic_mult = self.calculate_kinetic_multiplier(rpm)

    conductivity_mult = self.calculate_plasma_conductivity(enhanced) / self.params.conductivity_base
    rebco_mult = enhanced.get('rebco_stator_harvest', 1.0) * 2.0
    mhd_mult = 1.0 + (enhanced.get('cascaded_mhd', 1.0) - 1.0) * 0.7

    thermal_mult = np.prod([enhanced.get(k,1.0) for k in ['thermal_regeneration','grid_heat_recycling'
        'plasma_heat_recycle','external_waste_heat','ai_optimization','ai_heat_grid_or

    total_power = base_power * kinetic_mult * conductivity_mult * thermal_mult * rebco_mult * mhd_mult
    if grid_colocation:

```

```

total_power *= 5.0 + (enhanced.get('ai_heat_grid_oracle',1.0)-1.0)*45.0

parasitic_reduction = max(1.5, enhanced.get('marx_duty_reduction',1) * enhanced.get('mhd_self_pump
    (enhanced.get('module_clustering',1)**0.5) * enhanced.get('shared_tank_st
    enhanced.get('swarm_resonance',1)**0.3 * enhanced.get('ai_adaptive_contrc

cop = total_power / (self.params.parasitic_power / parasitic_reduction)
return self.safety_envelope.enforce(cop, self.params)

def full_report(self):
    self.explain_operation()
    base_effects = {k:1.0 for k in self.amplifiers}
    print("=== FULL SIMULATION REPORT ===")
    print(f"Base COP (no amplifiers): {self.calculate_cop(base_effects):.1f}x")
    print(f"Steady-State COP (all 38): {self.calculate_cop(self.amplifiers, False):.1f}x")
    print(f"Grid Co-location COP:      {self.calculate_cop(self.amplifiers, True):.1f}x")
    print(f"Carnot Efficiency:          {1 - self.params.T_cold/self.params.T_hot:.1%}")
    print("\nAll 38 amplifiers successfully integrated and active.")

# ===== EXECUTION =====
if __name__ == "__main__":
    sim = SCGHMHSimulator()
    sim.full_report()

```