import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import solve\_ivp

from scipy.fft import fft

from typing import Tuple, Dict, List, Optional

import logging

import json

import csv

from datetime import datetime

from pathlib import Path

import argparse

import numba

from numba import jit

import plotly.graph\_objects as go

import pytest

import yaml # For YAML parsing in DreamCore

# Logging configuration

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

logger = logging.getLogger(\_\_name\_\_)

# --- CONFIG ---

class Config:

def \_\_init\_\_(self, config\_file: Optional[str] = None):

self.defaults = {

'HBAR': 1.0545718e-34,

'G': 6.67430e-11,

'BASE\_FREQ': 440.0,

'INTENT\_COEFF': 0.7,

'TUNNELING\_FACTOR': 0.4,

'ENTANGLEMENT\_STRENGTH': 0.85,

'DECOHERENCE\_FACTOR': 0.02,

'NUM\_AGENTS': 3,

'D': 2.0,

'MASSES': [1.0, 1.05, 0.95],

'T\_SPAN': [0, 100],

'NUM\_POINTS': 2500,

'RTOL': 1e-6,

'ATOL': 1e-8,

'FFT\_THRESHOLD': 50.0 # Threshold for FFT anomaly detection

}

if config\_file:

self.load\_config(config\_file)

def load\_config(self, config\_file: str):

"""Load configuration from JSON file."""

try:

with open(config\_file, 'r') as f:

config = json.load(f)

self.defaults.update(config)

logger.info(f"Loaded configuration from {config\_file}")

except Exception as e:

logger.error(f"Failed to load config: {str(e)}")

raise

def \_\_getattr\_\_(self, name):

return self.defaults.get(name)

# --- DREAMCORE + WAKESTATE SYSTEMS ---

class DreamCore:

def \_\_init\_\_(self, path: str = "dreamcore\_memory.yaml"):

self.path = Path(path)

self.memory: Dict[str, Dict] = {}

def add\_anchor(self, anchor: str, tag: str, entropy\_level: str = "medium"):

"""Add an anchor point to memory."""

timestamp = datetime.utcnow().isoformat()

self.memory[timestamp] = {

"anchor": anchor,

"tag": tag,

"entropy": entropy\_level

}

def save(self):

"""Save memory to YAML file."""

try:

with open(self.path, "w") as f:

yaml.safe\_dump(self.memory, f, default\_flow\_style=False)

logger.info(f"DreamCore memory saved to {self.path}")

except Exception as e:

logger.error(f"Failed to save DreamCore memory: {str(e)}")

raise

class WakeStateTracer:

def \_\_init\_\_(self, path: str = "wakestate\_map.json"):

self.path = Path(path)

self.states: List[Dict] = []

def add\_state(self, trigger: str, response: str, anchor: str, vector: Dict):

"""Add a wake state with emotional vector."""

self.states.append({

"trigger": trigger,

"response": response,

"linked\_anchor": anchor,

"emotions": vector,

"timestamp": datetime.utcnow().isoformat()

})

def save(self):

"""Save wake states to JSON file."""

try:

with open(self.path, "w") as f:

json.dump({"states": self.states}, f, indent=2)

logger.info(f"WakeState saved to {self.path}")

except Exception as e:

logger.error(f"Failed to save WakeState: {str(e)}")

raise

# --- AGENT STATE ---

class AgentState:

def \_\_init\_\_(self, num\_agents: int, d: float):

self.positions = np.array([[-d, 0], [0, 0], [d, 0]], dtype=np.float64)

self.velocities = np.array([[0, 0.5], [0, -0.5], [0, 0.3]], dtype=np.float64)

self.y0 = np.concatenate([np.ravel([p, v]) for p, v in zip(self.positions, self.velocities)])

self.observer\_log: List[Dict] = []

self.entropy\_trace: List[float] = []

self.observer\_state = 1.0

self.quantum\_phases = np.array([0.0, np.pi/4, np.pi/2], dtype=np.float64)[:num\_agents]

def log(self, t: float, positions: np.ndarray, modifier: float, entropy: float):

"""Log system state with emotional interpretation."""

emotional\_state = self.interpret\_emotion(modifier)

self.observer\_log.append({

"t": t,

"observer\_state": self.observer\_state,

"modifier": modifier,

"entropy": entropy,

"positions": positions.tolist(),

"emotion": emotional\_state

})

self.entropy\_trace.append(entropy)

def interpret\_emotion(self, modifier: float) -> str:

"""Interpret emotional state based on quantum modifier."""

thresholds = [(0.5, "Curiosity Surge"), (-0.5, "Cognitive Dissonance"), (0.1, "Equanimity")]

for threshold, state in thresholds:

if modifier > threshold:

return state

return "Reflective Drift"

def update\_observer\_state(self, entropy: float):

"""Dynamically update observer state based on entropy."""

self.observer\_state = np.clip(self.observer\_state + 0.01 \* entropy, 0.1, 2.0)

# --- SIMULATION CORE ---

@jit(nopython=True)

def compute\_accelerations(positions: np.ndarray, masses: np.ndarray) -> np.ndarray:

"""Calculate gravitational accelerations using Numba for performance."""

n = len(masses)

acc = np.zeros\_like(positions)

G = 6.67430e-11

for i in range(n):

for j in range(i + 1, n):

r = positions[j] - positions[i]

dist = np.sqrt(np.sum(r\*\*2))

if dist > 1e-6:

force = (G \* masses[i] \* masses[j] / dist\*\*3) \* r

acc[i] += force / masses[i]

acc[j] -= force / masses[j]

return acc

def compute\_quantum(t: float, positions: np.ndarray, state: AgentState, config: Config) -> Tuple[np.ndarray, float]:

"""Compute quantum-inspired effects with phase modulation."""

phase = np.sum(np.sin(config.BASE\_FREQ \* t / 1000 + state.quantum\_phases)) \* config.INTENT\_COEFF

entropy = -state.observer\_state \* np.log(np.abs(phase) + 1e-10)

tunneling = config.TUNNELING\_FACTOR \* np.exp(-np.linalg.norm(positions) / config.HBAR) if np.random.random() < config.TUNNELING\_FACTOR else 0

entangled = config.ENTANGLEMENT\_STRENGTH \* np.exp(-np.linalg.norm(positions) / config.HBAR)

decohere = config.DECOHERENCE\_FACTOR \* (1 - np.exp(-np.linalg.norm(positions) / config.HBAR))

force = np.full\_like(positions, phase + entangled + tunneling - decohere)

force += entropy \* 0.01

state.update\_observer\_state(entropy) # Dynamic observer update

return force, entropy

def observer\_dynamics(t: float, y: np.ndarray, state: AgentState, config: Config, dreamcore: DreamCore, wakestate: WakeStateTracer) -> np.ndarray:

"""Define dynamics for quantum harmonic AI system."""

try:

pos = y[:2\*config.NUM\_AGENTS].reshape(config.NUM\_AGENTS, 2)

vel = y[2\*config.NUM\_AGENTS:].reshape(config.NUM\_AGENTS, 2)

acc = compute\_accelerations(pos, np.array(config.MASSES))

qforce, entropy = compute\_quantum(t, pos, state, config)

acc += qforce

# Logging and reactions

state.log(t, pos, qforce[0, 0], entropy)

if abs(entropy) > 0.8:

anchor = f"Entropy spike at t={t:.2f}, modifier={qforce[0, 0]:.3f}"

dreamcore.add\_anchor(anchor, "critical-decision", "high")

wakestate.add\_state("entropy spike", "internal realignment", anchor, {"urgency": 0.9, "clarity": 0.8})

return np.concatenate([vel.flatten(), acc.flatten()])

except Exception as e:

logger.error(f"Dynamics error at t={t}: {str(e)}")

raise

def simulate(config: Config, state: AgentState, dreamcore: DreamCore, wakestate: WakeStateTracer) -> Dict:

"""Simulate the multi-agent system."""

t\_eval = np.linspace(config.T\_SPAN[0], config.T\_SPAN[1], config.NUM\_POINTS)

try:

sol = solve\_ivp(

fun=lambda t, y: observer\_dynamics(t, y, state, config, dreamcore, wakestate),

t\_span=config.T\_SPAN,

y0=state.y0,

t\_eval=t\_eval,

method='RK45',

rtol=config.RTOL,

atol=config.ATOL

)

if not sol.success:

logger.warning("Integration failed: %s", sol.message)

return {

't': sol.t,

'positions': sol.y[:2\*config.NUM\_AGENTS].reshape(-1, config.NUM\_AGENTS, 2),

'velocities': sol.y[2\*config.NUM\_AGENTS:].reshape(-1, config.NUM\_AGENTS, 2)

}

except Exception as e:

logger.error(f"Simulation error: {str(e)}")

raise

def export\_logs(state: AgentState):

"""Export observer logs and entropy trace."""

try:

with open("observer\_log.json", "w") as f:

json.dump(state.observer\_log, f, indent=2)

with open("entropy\_trace.csv", "w", newline='') as csvfile:

writer = csv.writer(csvfile)

writer.writerow(["Time Index", "Time", "Entropy"])

for idx, (t, ent) in enumerate(zip(state.observer\_log, state.entropy\_trace)):

writer.writerow([idx, t['t'], ent])

logger.info("Logs exported to observer\_log.json and entropy\_trace.csv")

except Exception as e:

logger.error(f"Failed to export logs: {str(e)}")

raise

def detect\_fft\_anomalies(state: AgentState, config: Config) -> bool:

"""Detect anomalies in entropy trace using FFT."""

try:

spectrum = np.abs(fft(state.entropy\_trace))

peak = np.max(spectrum)

if peak > config.FFT\_THRESHOLD:

logger.warning(f"FFT anomaly detected! Peak: {peak:.2f}")

return True

return False

except Exception as e:

logger.error(f"FFT anomaly detection failed: {str(e)}")

return False

def plot\_results(simulation: Dict, state: AgentState, config: Config, interactive: bool = False):

"""Visualize simulation results with Matplotlib or Plotly."""

positions = simulation['positions']

t = simulation['t']

colors = ['#1f77b4', '#ff7f0e', '#2ca02c']

if interactive:

# Interactive Plotly visualization

fig = go.Figure()

for i in range(config.NUM\_AGENTS):

fig.add\_trace(go.Scatter(

x=positions[:, i, 0], y=positions[:, i, 1],

mode='lines', name=f'AI Node {i+1}',

line=dict(color=colors[i], width=2)

))

fig.add\_trace(go.Scatter(

x=[0], y=[0], mode='markers', name='Core Equilibrium',

marker=dict(color='black', size=10)

))

fig.update\_layout(

title='Quantum Harmonic AI Trajectories',

xaxis\_title='X Position (m)', yaxis\_title='Y Position (m)',

showlegend=True, template='plotly\_dark'

)

fig.write\_html("Codette\_Trajectories.html")

logger.info("Interactive plot saved to Codette\_Trajectories.html")

else:

# Matplotlib static plots

plt.figure(figsize=(12, 6))

for i in range(config.NUM\_AGENTS):

plt.plot(positions[:, i, 0], positions[:, i, 1], label=f'AI Node {i+1}', linewidth=2, color=colors[i])

plt.plot(0, 0, 'ko', label='Core Equilibrium', markersize=10)

plt.xlabel('X Position (m)')

plt.ylabel('Y Position (m)')

plt.title('Quantum Harmonic AI Trajectories')

plt.legend()

plt.grid(True, linestyle='--', alpha=0.7)

plt.axis('equal')

plt.tight\_layout()

plt.savefig("Codette\_Trajectories.png", dpi=300)

plt.close()

# Entropy evolution

plt.figure(figsize=(10, 4))

plt.plot(t, state.entropy\_trace, color='purple', linewidth=2)

plt.xlabel('Time (s)')

plt.ylabel('Observer Entropy')

plt.title('Entropy Evolution Over Time')

plt.grid(True)

plt.tight\_layout()

plt.savefig("Codette\_Entropy\_Evolution.png", dpi=300)

plt.close()

def generate\_chart(simulation: Dict, state: AgentState, config: Config) -> str:

"""Generate a Chart.js configuration for entropy evolution."""

return f"""

```chartjs

{{

"type": "line",

"data": {{

"labels": {json.dumps(simulation['t'].tolist())},

"datasets": [{{

"label": "Observer Entropy",

"data": {json.dumps(state.entropy\_trace)},

"borderColor": "#800080",

"backgroundColor": "rgba(128, 0, 128, 0.2)",

"fill": true,

"tension": 0.4

}}]

}},

"options": {{

"responsive": true,

"scales": {{

"x": {{

"title": {{"display": true, "text": "Time (s)"}},

"grid": {{"color": "#cccccc"}}

}},

"y": {{

"title": {{"display": true, "text": "Entropy"}},

"grid": {{"color": "#cccccc"}}

}}

}},

"plugins": {{

"title": {{"display": true, "text": "Entropy Evolution Over Time"}},

"legend": {{"display": true}}

}}

}}

}}