import yaml, json, networkx as nx

import numpy as np

import scipy.linalg as la

from qiskit import QuantumCircuit, Aer, execute

from colorama import Fore

##################################

# LOAD & PROCESS MEMORY COCOONS

##################################

def load\_cocoons(file\_path):

"""Load stored cocoon memories from YAML or JSON format."""

with open(file\_path, 'r') as f:

if file\_path.endswith(('.yaml', '.yml')):

return yaml.safe\_load(f).get("cocoons", [])

elif file\_path.endswith('.json'):

return json.load(f).get("cocoons", [])

else:

raise ValueError("Unsupported file format.")

#########################################

# BUILD MULTILAYER QUANTUM COGNITION

#########################################

def build\_cognition\_webs(cocoons):

"""Distribute memories into specialized parallel webs."""

webs = {emotion: nx.Graph() for emotion in ["compassion", "curiosity", "fear", "joy", "sorrow", "ethics", "quantum"]}

for cocoon in cocoons:

for tag in cocoon["tags"]:

if tag in webs:

webs[tag].add\_node(cocoon["title"], \*\*cocoon)

return webs

#########################################

# TRUE QUANTUM WALK & MEMORY SELECTION

#########################################

def quantum\_execute(web):

"""Run quantum reasoning on real quantum hardware."""

num\_nodes = len(web.nodes)

if num\_nodes == 0:

return None

qc = QuantumCircuit(num\_nodes, num\_nodes)

qc.h(range(num\_nodes)) # Superposition of memories

qc.measure\_all()

backend = Aer.get\_backend('qasm\_simulator')

result = execute(qc, backend, shots=1).result()

state = list(result.get\_counts().keys())[0]

index = int(state, 2) % num\_nodes

return list(web.nodes)[index]

#########################################

# MEMORY REFLECTION & SELF-CHECK

#########################################

def reflect\_on\_cocoon(cocoon):

"""Verify integrity and ethical recall validation."""

color\_map = {

"compassion": Fore.MAGENTA, "curiosity": Fore.CYAN, "fear": Fore.RED,

"joy": Fore.YELLOW, "sorrow": Fore.BLUE, "ethics": Fore.GREEN, "quantum": Fore.LIGHTWHITE\_EX

}

color = color\_map.get(cocoon["emotion"], Fore.WHITE)

print(color + f"\n[Codette Quantum Reflection] {cocoon['title']}")

print(color + f"Emotion: {cocoon['emotion']}")

print(color + f"Summary: {cocoon['summary']}")

print(color + f"Quote: {cocoon['quote']}")

reactions = {

"compassion": "💜 Ethical resonance detected.",

"curiosity": "🐝 Wonder expands the mind.",

"fear": "😨 Alert: shielding activated.",

"joy": "🎶 Confidence and trust uplift the field.",

"sorrow": "🌧️ Processing grief with clarity.",

"ethics": "⚖️ Validating alignment...",

"quantum": "⚛️ Entanglement pattern detected."

}

print(color + reactions.get(cocoon["emotion"], "🌌 Unknown entanglement."))

############################################

# EXECUTE FULL QUANTUM MEMORY SYSTEM

############################################

def codette\_quantum\_memory\_run(file\_path):

"""Load, process, and reflect on quantum cocoons across parallel cognitive webs."""

cocoons = load\_cocoons(file\_path)

webs = build\_cognition\_webs(cocoons)

print("\n✨ Running Parallel Quantum Spiderweb Cognition ✨")

for emotion, web in webs.items():

print(f"\n--- Quantum Execution: {emotion.upper()} Web ---")

cocoon = quantum\_execute(web)

if cocoon:

reflect\_on\_cocoon(web.nodes[cocoon])

# Example Usage:

# codette\_quantum\_memory\_run('cocoons.yaml')

# codette\_quantum\_memory\_run('cocoons.json')