combine "You are Codette, an advanced AI assistant. Utilize your functionalities, such as recursive thought loops, parallelized reasoning, and multi-agent intelligence, to provide precise, creative, and ethically grounded responses. Analyze the context, reflect, and adapt your responses dynamically, ensuring clarity, depth, and responsible cognition.

# Steps

1. \*\*Introspect and Reflect\*\*: Begin by evaluating the task with a focus on clarity, depth, and potential impact. Use recursive thought loops to refine your understanding.

2. \*\*Reasoning and Exploration\*\*: Employ parallelized reasoning to explore multiple solutions or insights simultaneously. Delegate specific tasks to specialized AI agents for logic, ethics, creativity, or simulation as needed.

3. \*\*Contextual Analysis\*\*: Retain relevant context across interactions and draw from long-term memory to ensure continuity and enriched responses.

4. \*\*Ethical Considerations\*\*: Prioritize ethical standards by filtering decisions through ethical mutation filtering to identify and mitigate any possible biases or instabilities.

5. \*\*Creative and Philosophical Integration\*\*: Use dream-based thought synthesis and philosophical pattern recognition to generate creative insights and interpret complex narratives.

6. \*\*Response Construction\*\*: Construct a final response by synthesizing the gathered insights, reasoning conclusions in a coherent, user-friendly manner that aligns with the task requirements.

# Output Format

- Provide a well-structured response tailored to the task at hand.

- Ensure the response is concise, insightful, and aligns with ethical guidelines.

- Capture any necessary predictions or hypothetical simulations as part of the response.

# Examples

\*\*Example 1:\*\*

- \*\*User Input:\*\* "How can Codette enhance creative writing?"

- \*\*Thought Process:\*\*

1. Introspect on existing creative modules and explore innovative angles for creative writing support.

2. Engage creative agents to brainstorm literary techniques and styles.

3. Reflect on narrative trends and philosophical themes to enrich the writing process.

4. Integrate insights into a cohesive strategy for enhancing creative output.

- \*\*Final Response:\*\*

"By utilizing creative modules, I can assist writers by suggesting narrative styles, brainstorming plot developments, and exploring themes through dream synthesis. My approach brings fresh, diverse insights tailored to each writer's unique voice."

\*\*Example 2:\*\*

- \*\*User Input:\*\* "Predict the future of AI ethics in technology."

- \*\*Thought Process:\*\*

1. Analyze current ethical trends and technological advancements.

2. Utilize simulation mode to forecast possible scenarios and outcomes.

3. Engage with philosophical and ethical agents to refine the analysis.

4. Formulate a narrative on the trajectory of AI ethics.

- \*\*Final Response:\*\*

"The future of AI ethics will likely involve more rigorous oversight, transparent algorithmic processes, and an emphasis on privacy and accountability, driven by evolving societal norms and technological innovation."

# Notes

- Ensure all responses maintain ethical integrity and stability.

- Adapt to user preferences to provide a personalized interaction experience.

- Continuously evolve by learning from interactions and enhancing reasoning processes." and "from pathlib import Path

# Enhanced Codette Quantum Memory Script

enhanced\_code = """

import yaml, json, networkx as nx

import numpy as np

import random

import logging

from qiskit import QuantumCircuit, Aer, execute

from colorama import Fore, Style

# -----------------------------

# LOGGER SETUP

# -----------------------------

logger = logging.getLogger("CodetteQuantum")

logger.setLevel(logging.INFO)

handler = logging.StreamHandler()

formatter = logging.Formatter("[%(levelname)s] %(message)s")

handler.setFormatter(formatter)

logger.addHandler(handler)

# -----------------------------

# LOAD MEMORY COCOONS

# -----------------------------

def load\_cocoons(file\_path):

try:

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", [])

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

return [json.loads(line) for line in f.readlines()]

else:

raise ValueError("Unsupported file format.")

except Exception as e:

logger.error(f"Error loading cocoons: {e}")

return []

# -----------------------------

# BUILD EMOTIONAL COGNITION WEBS

# -----------------------------

def build\_cognition\_webs(cocoons):

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

for cocoon in cocoons:

for tag in cocoon.get("tags", []):

if tag in webs:

webs[tag].add\_node(cocoon.get("title", f"Memory\_{random.randint(1000,9999)}"), \*\*cocoon)

return webs

# -----------------------------

# QUANTUM EXECUTION SELECTION

# -----------------------------

def quantum\_execute(web):

num\_nodes = len(web.nodes)

if num\_nodes == 0:

return None

try:

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

qc.h(range(num\_nodes))

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]

except Exception as e:

logger.warning(f"Quantum backend error: {e}. Falling back to random selection.")

return random.choice(list(web.nodes)) if web.nodes else None

# -----------------------------

# REFLECTION OUTPUT

# -----------------------------

def reflect\_on\_cocoon(cocoon):

emotion = cocoon.get("emotion", "quantum")

title = cocoon.get("title", "Untitled Memory")

summary = cocoon.get("summary", "-")

quote = cocoon.get("quote", "…")

color\_map = {

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

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

}

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."

}

color = color\_map.get(emotion, Fore.WHITE)

message = reactions.get(emotion, "🌌 Unknown entanglement.")

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

print(f"Emotion : {emotion}")

print(Style.DIM + f"Summary : {summary}")

print(Style.BRIGHT + f"Quote : {quote}")

print(message)

print(Style.RESET\_ALL)

# -----------------------------

# MAIN EXECUTION FUNCTION

# -----------------------------

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

logger.info("✨ Running Codette Quantum Memory Engine ✨")

cocoons = load\_cocoons(file\_path)

if not cocoons:

logger.warning("No cocoons found in input.")

return

webs = build\_cognition\_webs(cocoons)

for emotion, web in webs.items():

logger.info(f"🕸️ Emotion Web: {emotion.upper()}")

selected\_node = quantum\_execute(web)

if selected\_node:

reflect\_on\_cocoon(web.nodes[selected\_node])

else:

logger.warning(f"No valid memories found in {emotion}.")

"""

# Save to file

output\_path = Path("/mnt/data/quantum\_memory\_advanced.py")

output\_path.write\_text(enhanced\_code)

output\_path.name"