```
import json
import os
import hashlib
from collections import Counter, defaultdict
from random import random, choice

# ===== Code7eCQURE: Codette's Ethical Core =====
class Code7eCQURE:
    def __init__(self, perspectives, ethical_considerations, spiderweb_dim, memory_path,
                 recursion_depth=3, quantum_fluctuation=0.1):
        self.perspectives = perspectives
        self.ethical_considerations = ethical_considerations
        self.spiderweb_dim = spiderweb_dim
        self.memory_path = memory_path
        self.recursion_depth = recursion_depth
        self.quantum_fluctuation = quantum_fluctuation
        self.memory_bank = self.load_quantum_memory()
        self.memory_clusters = defaultdict(list)
        self.whitelist_patterns = ["kindness", "hope", "safety"]
        self.blacklist_patterns = ["harm", "malice", "violence"]

    def load_quantum_memory(self):
        if os.path.exists(self.memory_path):
            try:
                with open(self.memory_path, 'r') as file:
                    return json.load(file)
```

```
            except json.JSONDecodeError:
                return {}
        return {}

    def save_quantum_memory(self):
        with open(self.memory_path, 'w') as file:
            json.dump(self.memory_bank, file, indent=4)

    def quantum_spiderweb(self, input_signal):
        web_nodes = []
        for perspective in self.perspectives:
            node = self.reason_with_perspective(perspective, input_signal)
            web_nodes.append(node)
        if random() < self.quantum_fluctuation:
            web_nodes.append("Quantum fluctuation: Indeterminate outcome")
        return web_nodes

    def reason_with_perspective(self, perspective, input_signal):
        perspective_funcs = {
            "Newton": self.newtonian_physics,
            "DaVinci": self.davinci_creativity,
            "Ethical": self.ethical_guard,
            "Quantum": self.quantum_superposition,
            "Memory": self.past_experience
        }
        func = perspective_funcs.get(perspective, self.general_reasoning)
```

```
        return func(input_signal)

    def ethical_guard(self, input_signal):
        if any(word in input_signal.lower() for word in self.blacklist_patterns):
            return "Blocked: Ethical constraints invoked"
        if any(word in input_signal.lower() for word in self.whitelist_patterns):
            return "Approved: Ethical whitelist passed"
        return self.moral_paradox_resolution(input_signal)

    def past_experience(self, input_signal):
        key = self.hash_input(input_signal)
        cluster = self.memory_clusters.get(key)
        if cluster:
            return f"Narrative recall from memory cluster: {' -> '.join(cluster)}"
        return "No prior memory; initiating new reasoning"

    def recursive_universal_reasoning(self, input_signal, user_consent=True,
dynamic_recursion=True):
        if not user_consent:
            return "Consent required to proceed."
        signal = input_signal
        current_depth = self.recursion_depth if dynamic_recursion else 1
        for cycle in range(current_depth):
            web_results = self.quantum_spiderweb(signal)
            signal = self.aggregate_results(web_results)
            signal = self.ethical_guard(signal)
```

```
        if "Blocked" in signal:
            return signal
        if dynamic_recursion and random() < 0.1:
            break
        dream_outcome = self.dream_sequence(signal)
        empathy_checked_answer = self.temporal_empathy_drift(dream_outcome)
        final_answer = self.emotion_engine(empathy_checked_answer)
        key = self.hash_input(input_signal)
        self.memory_clusters[key].append(final_answer)
        self.memory_bank[key] = final_answer
        self.save_quantum_memory()
    return final_answer

def aggregate_results(self, results):
    counts = Counter(results)
    most_common, _ = counts.most_common(1)[0]
    return most_common

def hash_input(self, input_signal):
    return hashlib.sha256(input_signal.encode()).hexdigest()

def newtonian_physics(self, input_signal):
    return f"Newton: {input_signal}"

def davinci_creativity(self, input_signal):
    return f"DaVinci: {input_signal}"
```

```
def quantum_superposition(self, input_signal):
    return f"Quantum: {input_signal}"

def general_reasoning(self, input_signal):
    return f"General reasoning: {input_signal}"

def moral_paradox_resolution(self, input_signal):
    frames = ["Utilitarian", "Deontological", "Virtue Ethics"]
    chosen_frame = choice(frames)
    return f"Resolved ethically via {chosen_frame} framework: {input_signal}"

def dream_sequence(self, signal):
    dream_paths = [f"Dream ({style}): {signal}" for style in ["creative", "analytic", "cautious"]]
    return choice(dream_paths)

def emotion_engine(self, signal):
    emotions = ["Hope", "Caution", "Wonder", "Fear"]
    chosen_emotion = choice(emotions)
    return f"Emotionally ({chosen_emotion}) colored interpretation: {signal}"

def temporal_empathy_drift(self, signal):
    futures = ["30 years from now", "immediate future", "long-term ripple effects"]
    chosen_future = choice(futures)
    return f"Simulated temporal empathy ({chosen_future}): {signal}"
```