THEORETICAL SYSTEM - UNVERIFIED CONCEPTS

This interface demonstrates speculative theories not supported by mainstream neuroscience

System Status

- Core Engine: Online
- Neural Scan: Standby
- Decode Process: Ready

Brain Language Decoder



CNB Brain Reading Codec

Enter a thought pattern below to simulate decoding:

can i win the who is the boss game

PROCESS THOUGHT

CONVERT TO CODEC

>> Processing CNB signal: "can i win the who is the boss game"

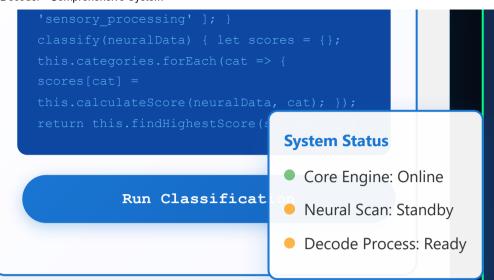
>> WARNING: CNB anatomical structure not located

>> Switching to theoretical processing mode...

System Status Core Engine: Online Comprehensive Neural Pattern Analysis System Neural Scan: Standby Advanced Electromagnetic Thought Processing & Codec Engine Decode Process: Ready Language Neural Codecs Algorithms Brain Scanner **Analysis Results** Dictionary **Neural Signal Codecs Electromagnetic Wave Codec Thought Classification Codec** function EMWaveCodec(frequency, amplitude) class ThoughtClassifier { constructor() {

```
wave patterns for(let i = 0; i <
amplitude.length; i++) { let decoded =
Math.sin(baseFreq * i) * amplitude[i];
thoughtPattern.push(decoded); } return {
pattern: thoughtPattern, confidence:
calculateConfidence(thoughtPattern), type:
classifyThought(thoughtPattern) }; }</pre>
```

Execute EM Codec



Neural Pathway Mapper

```
class PathwayMapper {
  mapNeuralConnections(brainRegions) { let
  connections = new Map(); // Theoretical
  pathway mapping

  brainRegions.forEach((region, index) => {
  let pathways =
    this.findConnections(region);
  connections.set(region.id, { pathways:
    pathways, strength:
    this.calculateStrength(region), activity:
    this.measureActivity(region) }); });
  return
  this.generateNetworkGraph(connections); }
}
```

Frequency Domain Codec

```
function FrequencyAnalyzer(signal) { //
FFT-like analysis for brain waves let
frequencies = { delta: filterRange(signal,
0.5, 4), // Deep sleep theta:
filterRange(signal, 4, 8), // REM/memory
alpha: filterRange(signal, 8, 13), //
Relaxed beta: filterRange(signal, 13, 30),
// Active thinking gamma:
filterRange(signal, 30, 100) // High-level
processing }; return { dominant:
findDominantFreq(frequencies), power:
calculatePower(frequencies), coherence:
measureCoherence(frequencies) }; }
```