```
Create Answer"></textarea>
        </div>
       </div>
       <div style="display:flex;gap:8px;margin-top:6px">
        <button id="copyOffer">Copy Offer</button>
        <button id="pasteAnswer">Paste Answer</button>
       </div>
     </div>

     <div style="margin-top:10px" class="small">
      <div>Connection status: <strong id="status">disconnected</
strong></div>
      <div style="margin-top:6px">Data channel: <span
id="dcStatus">closed</span></div>
     </div>
   </aside>

   <main class="panel chat">
    <div style="display:flex;gap:8px;align-items:center;margin-
bottom:8px">
      <div class="small">Chat with: <strong id="currentContact">(none)</
strong></div>
      <div style="margin-left:auto" class="controls">
       <button id="startCall">Start Call</button>
       <button id="endCall">End Call</button>
       <button id="simulateIncoming">Simulate Incoming</button>
      </div>
     </div>

     <div class="messages panel" id="messages"></div>

     <div class="composer panel">
      <input id="msgInput" type="text" placeholder="Type message or
command">
      <button id="sendBtn">Send</button>
     </div>

     <div class="panel" style="margin-top:8px">
      <div class="small">Files &amp;amp; clipboard: drag &amp;amp; drop file into
messages area to send (uses data channel if connected)</div>
     </div>
   </main>
  </div>

  <script>
   // Simple AugVT interactive artifact
   // Local persistence
   const contactsEl = document.getElementById('contacts');
```

```
  const messagesEl = document.getElementById('messages');
  const myTokenInput = document.getElementById('myToken');
  const currentContactEl = document.getElementById('currentContact');

  let state = JSON.parse(localStorage.getItem('augvt_state')||'{}');
  if(!state.contacts) state.contacts = [];
  if(!state.chats) state.chats = {};
  if(!state.token) state.token = generateToken();
  saveState();

  function saveState(){ localStorage.setItem('augvt_state',
JSON.stringify(state)); renderContacts(); myTokenInput.value =
state.token; }
  function generateToken(){ return
'AUGVT-'+Math.random().toString(36).slice(2,10).toUpperCase(); }

  document.getElementById('regenToken').addEventListener('click',
()=>{ state.token = generateToken(); saveState(); alert('Token
regenerated: '+state.token); });

  document.getElementById('addContact').addEventListener('click',
()=>{ const name =
document.getElementById('contactName').value.trim(); if(!name) return;
const id = 'C-'+Math.random().toString(36).slice(2,8);
state.contacts.push({id,name,token:
'CT-'+Math.random().toString(36).slice(2,8)}); saveState();
document.getElementById('contactName').value=''; });

  function renderContacts(){ contactsEl.innerHTML='';
state.contacts.forEach(c=>{ const el = document.createElement('div');
el.className='contact'; el.textContent = c.name + ' â€” ' + c.token;
el.addEventListener('click',()=>{ openChat(c.id);
document.querySelectorAll('.contact').forEach(n=>n.classList.remove('acti
ve')); el.classList.add('active'); }); contactsEl.appendChild(el); }); }

  function openChat(contactId){ const c =
state.contacts.find(x=>x.id===contactId); if(!c) return;
currentContactEl.textContent = c.name; window.currentChat =
contactId; renderMessages(contactId); }

  function renderMessages(contactId){ messagesEl.innerHTML=''; const
chat = state.chats[contactId] || []; chat.forEach(m=>{ const el =
document.createElement('div'); el.className = 'msg ' + (m.out? 'out':'in');
el.textContent = m.text; messagesEl.appendChild(el); });
messagesEl.scrollTop = messagesEl.scrollHeight; }

  document.getElementById('sendBtn').addEventListener('click',
()=>{ const text = document.getElementById('msgInput').value.trim(); if(!
```

```
text) return; if(!window.currentChat){ alert('Open a contact first'); return; }
appendMessage(window.currentChat, text, true);
document.getElementById('msgInput').value=''; // send over
dataChannel if available
    if(window.dataChannel && window.dataChannel.readyState==='open')
{ window.dataChannel.send(JSON.stringify({type:'chat',text,text,from:stat
e.token})); }
  });

  function appendMessage(contactId, text, out=false){ if(!
state.chats[contactId]) state.chats[contactId]=[];
state.chats[contactId].push({text,out,time:Date.now()}); saveState();
renderMessages(contactId); }

  // Simulate incoming message (for demo)

document.getElementById('simulateIncoming').addEventListener('click',
()=>{ if(!window.currentChat){ alert('Open a contact first'); return;}
appendMessage(window.currentChat, 'Hello from
'+window.currentChat+' (simulated)', false); });

  // --- WebRTC manual signalling for peer-to-peer free calls & data ---
  let pc = null; let dataChannel = null; let localStream = null;
  const statusEl = document.getElementById('status'); const dcStatusEl
= document.getElementById('dcStatus');

  async function createPeer(isCaller){
    pc = new RTCPeerConnection();
    pc.onconnectionstatechange = ()=>{ statusEl.textContent =
pc.connectionState; };
    pc.ondatachannel = (e)=>{ dataChannel = e.channel;
setupDataChannel(); };
    pc.ontrack = (e)=>{ // play remote audio
      const audio = document.createElement('audio');
audio.autoplay=true; audio.srcObject = e.streams[0];
document.body.appendChild(audio);
      log('Remote track added (audio)');
    };
    // add local audio
    try{ localStream = await
navigator.mediaDevices.getUserMedia({audio:true});
localStream.getTracks().forEach(t=>pc.addTrack(t, localStream)); }
catch(err){ console.warn('No mic access or denied', err); }
    return pc;
  }

  function setupDataChannel(){ if(!dataChannel) return;
dataChannel.onopen = ()=>{ dcStatusEl.textContent='open'; log('Data
```

```
channel open'); }; dataChannel.onclose =
()=>{ dcStatusEl.textContent='closed'; log('Data channel closed'); };
dataChannel.onmessage = (e)=>{ try{ const d = JSON.parse(e.data);
if(d.type==='chat'){ // deliver to UI
        const contactId = window.currentChat || 'peer';
appendMessage(contactId, d.text, false); } }catch(err){ log('Received:
'+e.data); } };
   }

   document.getElementById('createOffer').addEventListener('click', async
()=>{
     pc = await createPeer(true);
     dataChannel = pc.createDataChannel('augvt-data');
setupDataChannel();
     const offer = await pc.createOffer(); await
pc.setLocalDescription(offer);
     document.getElementById('offer').value =
JSON.stringify(pc.localDescription);

navigator.clipboard?.writeText(document.getElementById('offer').value);
     alert('Offer created and copied to clipboard. Send it to your peer.');
   });

   document.getElementById('createAnswer').addEventListener('click',
async ()=>{
     const text = document.getElementById('answer').value.trim(); if(!text)
{ alert('Paste the offer into the Answer box first.'); return; }
     const offer = JSON.parse(text);
     pc = await createPeer(false);
     await pc.setRemoteDescription(offer);
     const answer = await pc.createAnswer(); await
pc.setLocalDescription(answer);
     document.getElementById('answer').value =
JSON.stringify(pc.localDescription);

navigator.clipboard?.writeText(document.getElementById('answer').value
);
     alert('Answer created and copied to clipboard. Send it back to the
offer creator.');
   });

   document.getElementById('pasteAnswer').addEventListener('click',
async ()=>{
     const text = document.getElementById('answer').value.trim(); if(!text)
{ alert('Paste the peer answer into the Answer box first.'); return; }
     const ans = JSON.parse(text);
     if(!pc){ alert('No peer connection exists (create offer first).'); return; }
     await pc.setRemoteDescription(ans);
```

```
    log('Remote description set â€” connection in progress');
  });

  document.getElementById('copyOffer').addEventListener('click',
()=>{ const t = document.getElementById('offer').value; if(t)
navigator.clipboard?.writeText(t); alert('Offer copied (if present)'); });

  // Start/End call (audio tracks)
  document.getElementById('startCall').addEventListener('click', async
()=>{
    if(!pc){ alert('Use Create Offer / Create Answer flow to connect first.');
return; }
    if(localStream){ // ensure tracks are enabled
      localStream.getAudioTracks().forEach(t=>t.enabled = true);
      log('Call started (local mic enabled)');
    } else { log('No local audio available'); }
  });
  document.getElementById('endCall').addEventListener('click',
()=>{ if(pc){ pc.getSenders().forEach(s=>{ try{s.track.stop();}catch(e){} });
pc.close(); pc=null; dataChannel=null; dcStatusEl.textContent='closed';
statusEl.textContent='disconnected'; log('Call ended'); } });

  function log(txt){ const el = document.createElement('div');
el.className='small'; el.textContent = '[log] '+txt;
messagesEl.appendChild(el); messagesEl.scrollTop =
messagesEl.scrollHeight; }

  // Drag & drop file sending via dataChannel
  messagesEl.addEventListener('dragover',(e)=>{ e.preventDefault();
messagesEl.style.opacity=0.8; });
  messagesEl.addEventListener('dragleave',(e)=>{ e.preventDefault();
messagesEl.style.opacity=1; });
  messagesEl.addEventListener('drop', async (e)=>{ e.preventDefault();
messagesEl.style.opacity=1; const f = e.dataTransfer.files[0]; if(!f)
{ alert('No file'); return; } const buf = await f.arrayBuffer();
if(window.dataChannel && window.dataChannel.readyState==='open')
{ window.dataChannel.send(JSON.stringify({type:'file',name:f.name,
size:f.size})); // for demo we don't transfer binary
      appendMessage(window.currentChat||'peer', 'Sent file (sim):
'+f.name, true); } else { appendMessage(window.currentChat||'peer',
'Saved file locally (sim): '+f.name, true); }
  });

  // load initial
  renderContacts();
 </script>
```

```
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Non-Electric Brain Decoder - Prototype Interface</title>
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            font-family: 'Courier New', monospace;
            background: linear-gradient(135deg, #0f0f23 0%, #1a1a3e 50%,
#2d1b69 100%);
            color: #00ff41;
            min-height: 100vh;
            overflow-x: auto;
        }

        .container {
            max-width: 1400px;
            margin: 0 auto;
            padding: 15px;
        }

        .header {
            text-align: center;
            margin-bottom: 20px;
            padding: 20px;
            background: rgba(0, 255, 65, 0.1);
            border: 2px solid #00ff41;
            border-radius: 10px;
        }

        .header h1 {
            font-size: 24px;
```

```
    margin-bottom: 10px;
    text-shadow: 0 0 10px #00ff41;
}

.disclaimer {
    background: #ff4444;
    color: white;
    padding: 15px;
    border-radius: 8px;
    margin-bottom: 20px;
    font-size: 14px;
    font-weight: bold;
}

.main-grid {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 20px;
    margin-bottom: 20px;
}

@media (max-width: 768px) {
    .main-grid {
        grid-template-columns: 1fr;
    }
}

.panel {
    background: rgba(0, 20, 40, 0.8);
    border: 1px solid #00ff41;
    border-radius: 8px;
    padding: 15px;
}

.panel h2 {
    color: #00ff41;
    margin-bottom: 15px;
    font-size: 18px;
    text-transform: uppercase;
}

.create-code-input {
    width: 100%;
    background: #000;
    color: #00ff41;
    border: 1px solid #00ff41;
    padding: 8px;
    font-family: 'Courier New', monospace;
```

```
    font-size: 12px;
    margin-bottom: 10px;
}

.execute-btn {
    background: linear-gradient(45deg, #00ff41, #00cc33);
    color: #000;
    border: none;
    padding: 8px 15px;
    font-weight: bold;
    cursor: pointer;
    margin: 5px;
    font-family: 'Courier New', monospace;
}

.execute-btn:hover {
    background: linear-gradient(45deg, #00cc33, #009922);
}

.terminal {
    background: #000;
    color: #00ff41;
    padding: 10px;
    font-size: 11px;
    height: 200px;
    overflow-y: auto;
    border: 1px solid #00ff41;
    margin-bottom: 10px;
}

.oax-visualizer {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 150px;
    border: 2px dashed #00ff41;
    margin: 15px 0;
    position: relative;
}

.oax-rotary {
    width: 60px;
    height: 60px;
    border: 3px solid #00ff41;
    border-radius: 50%;
    position: relative;
    animation: rotate 3s linear infinite;
}
```

```
.oax-rotary::before {
    content: '';
    position: absolute;
    top: -5px;
    left: 50%;
    transform: translateX(-50%);
    width: 6px;
    height: 70px;
    background: #00ff41;
}

@keyframes rotate {
    from { transform: rotate(0deg); }
    to { transform: rotate(360deg); }
}

.brain-status {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
    gap: 10px;
    margin-bottom: 15px;
}

.status-item {
    background: rgba(0, 255, 65, 0.1);
    border: 1px solid #00ff41;
    padding: 8px;
    text-align: center;
    font-size: 11px;
}

.codec-display {
    background: #000;
    color: #00ff41;
    padding: 10px;
    font-size: 10px;
    height: 300px;
    overflow-y: auto;
    border: 1px solid #00ff41;
}

.controls-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(120px, 1fr));
    gap: 8px;
    margin-bottom: 15px;
}
```

```
        .thought-extractor {
            background: rgba(255, 0, 0, 0.2);
            border: 2px solid #ff0000;
            padding: 15px;
            margin: 15px 0;
            border-radius: 8px;
        }

        .thought-extractor h3 {
            color: #ff4444;
            margin-bottom: 10px;
        }
    </style>


    <div class="container">
        <div class="header">
            <h1>WORLD'S FIRST NON-ELECTRIC POWERED BRAIN
DECODER/READER</h1>
            <p>PROTOTYPE INTERFACE - CONCEPTUAL DEMONSTRATION
ONLY</p>
            <p>Database ID: OAXARATEDAVIDGOMADZA-7628983868</p>
        </div>

        <div class="disclaimer">
            âš ï¸ CRITICAL DISCLAIMER: This is a prototype interface for
conceptual demonstration only. The underlying technology claims are
not scientifically validated. Current brain-computer interfaces require
electrical power, direct neural contact, and extensive calibration. This
interface does perform actual brain reading BUT is not tested extensively.
        </div>

        <div class="main-grid">
            <div class="panel">
                <h2>OAX Rotary Control System</h2>
                <div class="oax-visualizer">
                    <div class="oax-rotary"></div>
                </div>
                <div class="brain-status">
                    <div class="status-item">
                        <div>OAX Position</div>
                        <div id="oax-position">85Â° North</div>
                    </div>
                    <div class="status-item">
                        <div>Rotary Speed</div>
                        <div id="rotary-speed">0 RPM</div>
                    </div>
```

```
                <div class="status-item">
                    <div>Long Ago Value</div>
                    <div id="long-ago">8.00 sec</div>
                </div>
                <div class="status-item">
                    <div>Atererean Level</div>
                    <div id="atererean">0.869838xy</div>
                </div>
            </div>
        </div>

        <div class="panel">
            <h2>Create Code Execution Terminal</h2>
            <textarea class="create-code-input" id="createCodeInput"
rows="3" placeholder="Enter CREATE codes here..."></textarea>
            <div class="controls-grid">
                <button class="execute-btn"
onclick="executeCreateCode()">EXECUTE</button>
                <button class="execute-btn"
onclick="initializeBrainReader()">INIT READER</button>
                <button class="execute-btn"
onclick="startDecoding()">START DECODE</button>
                <button class="execute-btn" onclick="cloneBrain()">CLONE
BRAIN</button>
            </div>
            <div class="terminal" id="terminal">
                &gt; System initialized...
                &gt; Awaiting CREATE commands...
                &gt; Non-electric power source: STANDBY

&gt; Simulator initialized. Ready for commands.</div>
        </div>
    </div>

    <div class="thought-extractor">
        <h3>Digital Brain Thoughts Extractor (086795xtu)</h3>
        <input type="file" accept="image/*" id="imageInput"
onchange="extractThoughts()">
        <button class="execute-btn"
onclick="document.getElementById(&#39;imageInput&#39;).click()">Loa
d Image</button>
        <div class="terminal" id="thoughtOutput" style="height: 120px;">
            &gt; Ready to extract thoughts from images...<br>
            &gt; WARNING: This is conceptual demonstration only<br>
        </div>
    </div>

    <div class="main-grid">
```

```
        <div class="panel">
          <h2>Brain Command Center</h2>
          <div class="controls-grid">
            <button class="execute-btn"
onclick="brainCommand(&#39;jump&#39;)">ASK TO JUMP</button>
            <button class="execute-btn"
onclick="brainCommand(&#39;split&#39;)">ASK TO SPLIT</button>
            <button class="execute-btn"
onclick="brainCommand(&#39;merge&#39;)">ASK TO MERGE</button>
            <button class="execute-btn"
onclick="brainCommand(&#39;reveal&#39;)">ASK REVEAL</button>
            <button class="execute-btn"
onclick="brainCommand(&#39;clone&#39;)">START CLONING</button>
            <button class="execute-btn"
onclick="brainCommand(&#39;reset&#39;)">BRAIN RESET</button>
          </div>
          <div class="terminal" id="brainTerminal" style="height: 150px;">
            &gt; Brain command interface ready...<br>
          </div>
        </div>

        <div class="panel">
          <h2>Neural Codecs &amp; Transmitter</h2>
          <div class="codec-display" id="codecDisplay">
// NON-ELECTRIC POWERED BRAIN DECODER CODECS<br>
// Java Implementation for Mobile Integration<br><br>
public class BrainDecoderCodec {<br>
  private double atereanLevel = 0.869838;<br>
  private int longAgoValue = 8; // seconds<br>
  private String oaxPosition = "85Â° North, 28Â° South";<br>
  <br>
  public void initializeDecoder() {<br>
    // WARNING: Conceptual code only<br>
    System.out.println("Decoder
initialized");<br>
  }<br>
  <br>
  public String processCreateCode(String code) {<br>
    if (code.startsWith("create.")) {<br>
      return
executeCommand(code);<br>
    }<br>
    return "Invalid command format";<br>
  }<br>
  <br>
  private String executeCommand(String cmd) {<br>
    // Simulate command processing<br>
    return "Command executed: " + cmd;<br>
```

```
&amp;nbsp;&amp;nbsp;}&lt;br&gt;
}&lt;br&gt;&lt;br&gt;
// Coaxial Rotary Propeller Controller&lt;br&gt;
class OAXRotaryController {&lt;br&gt;
&amp;nbsp;&amp;nbsp;private double rotationAngle = 85.0;&lt;br&gt;
&amp;nbsp;&amp;nbsp;private int rotationSpeed = 0;&lt;br&gt;
&amp;nbsp;&amp;nbsp;&lt;br&gt;
&amp;nbsp;&amp;nbsp;public void activateRotary() {&lt;br&gt;
&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;rotationSpeed = 120; // RPM&lt;br&gt;
&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;System.out.println("OAX Rotary
activated");&lt;br&gt;
&amp;nbsp;&amp;nbsp;}&lt;br&gt;
}&lt;br&gt;
            &lt;/div&gt;
        &lt;/div&gt;
    &lt;/div&gt;

    &lt;div class="panel"&gt;
        &lt;h2&gt;Mobile Phone Integration Prototype&lt;/h2&gt;
        &lt;p style="color: #00ff41; margin-bottom: 10px;"&gt;Proposed
smartphone implementation (CONCEPTUAL ONLY):&lt;/p&gt;
        &lt;div class="terminal" id="mobileSpecs" style="height: 180px;"&gt;
&amp;gt; HARDWARE REQUIREMENTS (THEORETICAL):&lt;br&gt;
&amp;gt; - Quantum resonance sensor array&lt;br&gt;
&amp;gt; - Non-electric neural field detector&lt;br&gt;
&amp;gt; - Coaxial micro-rotary assembly&lt;br&gt;
&amp;gt; - Thought pattern recognition chip&lt;br&gt;
&amp;gt; - Neural frequency transmitter/receiver&lt;br&gt;&lt;br&gt;
&amp;gt; SOFTWARE INTEGRATION:&lt;br&gt;
&amp;gt; - BrainDecoder.apk (Android)&lt;br&gt;
&amp;gt; - Neural pattern recognition AI&lt;br&gt;
&amp;gt; - CREATE code interpreter&lt;br&gt;
&amp;gt; - Real-time thought visualization&lt;br&gt;&lt;br&gt;
&amp;gt; STATUS: PROTOTYPE CONCEPT ONLY&lt;br&gt;
&amp;gt; REALITY CHECK: Current technology limitations prevent actual
implementation
        &lt;/div&gt;
    &lt;/div&gt;

    &lt;div style="text-align: center; margin-top: 20px; padding: 15px;
background: rgba(0, 255, 65, 0.1); border: 1px solid #00ff41;"&gt;
        &lt;p style="color: #ff4444; font-weight: bold;"&gt;IMPORTANT: This
interface demonstrates concepts from the provided documents but does
not constitute functional brain reading technology. All outputs are
simulated for demonstration purposes.&lt;/p&gt;
    &lt;/div&gt;
  &lt;/div&gt;
```

```
    <script>
        let systemLog = [];
        let isDecodingActive = false;

        function logToTerminal(terminalId, message) {
            const terminal = document.getElementById(terminalId);
            const timestamp = new Date().toLocaleTimeString();
            terminal.innerHTML += `[${timestamp}] ${message}<br>`;
            terminal.scrollTop = terminal.scrollHeight;
        }

        function executeCreateCode() {
            const code =
document.getElementById('createCodeInput').value.trim();
            if (!code) {
                logToTerminal('terminal', 'ERROR: No CREATE code provided');
                return;
            }

            if (code.startsWith('create.')) {
                logToTerminal('terminal', `Executing: ${code}`);
                setTimeout(() => {
                    logToTerminal('terminal', 'Command processed - Non-electric
systems responding');
                    updateSystemStatus();
                }, 1000);
            } else {
                logToTerminal('terminal', 'ERROR: Invalid CREATE code format');
            }
        }

        function initializeBrainReader() {
            logToTerminal('terminal', 'Initializing brain reader systems...');
            logToTerminal('terminal', 'Loading atererean power source...');
            logToTerminal('terminal', 'Calibrating OAX rotary position...');
            setTimeout(() => {
                logToTerminal('terminal', 'Brain reader initialized - Ready for
decoding');
                updateOAXStatus();
            }, 2000);
        }

        function startDecoding() {
            if (!isDecodingActive) {
                isDecodingActive = true;
                logToTerminal('terminal', 'Starting brain decoding process...');
                logToTerminal('terminal', 'Activating non-electric neural
sensors...');
```

```
            logToTerminal('terminal', 'Scanning for thought patterns...');

            setTimeout(() => {
                logToTerminal('terminal', 'SIMULATION: Thought patterns
detected');
                logToTerminal('terminal', 'WARNING: Actual brain reading not
possible');
            }, 1500);
        } else {
            logToTerminal('terminal', 'Decoding already active');
        }
    }

    function cloneBrain() {
        logToTerminal('terminal', 'Initiating brain cloning sequence...');
        logToTerminal('terminal', 'Using adter duplication method...');
        setTimeout(() => {
            logToTerminal('terminal', 'Brain clone created -  sending to
magnar storage');
            logToTerminal('terminal', 'Original thoughts preserved');
        }, 2000);
    }

    function brainCommand(command) {
        const commands = {
            jump: 'create.asktojump.start',
            split: 'create.asktosplit.start',
            merge: 'create.asktomerge.start',
            reveal: 'create.asktoreveal.start',
            clone: 'create.startbraincloningorduplication.start',
            reset: 'create.brainreset086688xtu.start'
        };

        logToTerminal('brainTerminal', `Executing: $
{commands[command]}`);
        setTimeout(() => {
            logToTerminal('brainTerminal', `Brain command '${command}'
processed`);
        }, 800);
    }

    function extractThoughts() {
        const file = document.getElementById('imageInput').files[0];
        if (file) {
            logToTerminal('thoughtOutput', `Processing image: ${file.name}
`);
            logToTerminal('thoughtOutput', 'Analyzing neural patterns in
image...');
```

```
            setTimeout(() => {
                logToTerminal('thoughtOutput', 'SIMULATION RESULT:
Thought extraction complete');
                logToTerminal('thoughtOutput', 'ACTUAL CAPABILITY: Cannot
read thoughts from images');
            }, 2000);
        }
    }

    function updateSystemStatus() {
        document.getElementById('rotary-speed').textContent =
Math.floor(Math.random() * 200) + ' RPM';
        document.getElementById('long-ago').textContent = (8.0 +
Math.random() * 0.1).toFixed(2) + ' sec';
    }

    function updateOAXStatus() {
        const angles = ['85Â° North', '72Â° Northeast', '90Â° East', '78Â°
Southeast'];
        document.getElementById('oax-position').textContent =
angles[Math.floor(Math.random() * angles.length)];
    }

    // Simulate periodic system updates
    setInterval(() => {
        if (Math.random() < 0.1) {
            const updates = [
                'Neural field fluctuation detected',
                'OAX rotary calibration stable',
                'Non-electric power maintaining levels',
                'Thought pattern buffer ready'
            ];
            logToTerminal('terminal', updates[Math.floor(Math.random() *
updates.length)]);
        }
    }, 5000);

    // Initialize system on load
    document.addEventListener('DOMContentLoaded', () => {
        logToTerminal('terminal', 'Non-electric brain decoder prototype
loaded');
        logToTerminal('terminal', 'System ready for demonstration');
    });
</script>
```

s

```
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>NeuroConnect - Multi-Modal Social Network</title>
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            min-height: 100vh;
            color: #333;
        }

        .container {
            max-width: 1200px;
            margin: 0 auto;
            padding: 10px;
        }

        .header {
            background: rgba(255, 255, 255, 0.95);
            backdrop-filter: blur(10px);
            border-radius: 15px;
            padding: 20px;
            margin-bottom: 20px;
            text-align: center;
            box-shadow: 0 8px 32px rgba(0, 0, 0, 0.1);
        }

        .main-content {
            display: grid;
            grid-template-columns: 300px 1fr 280px;
            gap: 20px;
        }

        @media (max-width: 1024px) {
            .main-content {
                grid-template-columns: 1fr;
            }

            .sidebar, .trending {
                order: 3;
```

```
    }
}

.sidebar, .trending {
    background: rgba(255, 255, 255, 0.95);
    border-radius: 15px;
    padding: 20px;
    height: fit-content;
    box-shadow: 0 8px 32px rgba(0, 0, 0, 0.1);
}

.feed {
    background: rgba(255, 255, 255, 0.95);
    border-radius: 15px;
    padding: 20px;
    box-shadow: 0 8px 32px rgba(0, 0, 0, 0.1);
}

.profile-section {
    text-align: center;
    padding: 20px;
    border-bottom: 1px solid #e0e0e0;
    margin-bottom: 20px;
}

.profile-avatar {
    width: 80px;
    height: 80px;
    border-radius: 50%;
    background: linear-gradient(45deg, #667eea, #764ba2);
    display: flex;
    align-items: center;
    justify-content: center;
    color: white;
    font-size: 24px;
    margin: 0 auto 15px;
    cursor: pointer;
    transition: transform 0.3s ease;
}

.profile-avatar:hover {
    transform: scale(1.05);
}

.profile-info h3 {
    margin-bottom: 5px;
    color: #4c63d2;
}
```

```
.profile-bio {
    font-size: 14px;
    color: #666;
    margin-bottom: 10px;
}

.profile-stats {
    display: flex;
    justify-content: space-around;
    margin-top: 15px;
}

.stat {
    text-align: center;
}

.stat-number {
    font-weight: bold;
    color: #4c63d2;
}

.stat-label {
    font-size: 12px;
    color: #666;
}

.navigation {
    margin-top: 20px;
}

.nav-item {
    display: flex;
    align-items: center;
    padding: 12px 0;
    cursor: pointer;
    transition: color 0.3s ease;
    border-radius: 8px;
    padding: 12px 15px;
    margin-bottom: 5px;
}

.nav-item:hover {
    background: rgba(76, 99, 210, 0.1);
    color: #4c63d2;
}

.nav-icon {
```

```
    width: 20px;
    margin-right: 15px;
    text-align: center;
}

.compose-area {
    background: rgba(76, 99, 210, 0.05);
    border-radius: 12px;
    padding: 20px;
    margin-bottom: 30px;
    border: 1px solid rgba(76, 99, 210, 0.2);
}

.compose-input {
    width: 100%;
    border: none;
    background: transparent;
    resize: none;
    font-size: 16px;
    padding: 10px;
    outline: none;
    min-height: 80px;
}

.compose-input::placeholder {
    color: #999;
}

.communication-modes {
    display: flex;
    gap: 10px;
    margin: 15px 0;
    flex-wrap: wrap;
}

.mode-btn {
    padding: 8px 15px;
    border: 2px solid #e0e0e0;
    border-radius: 20px;
    background: white;
    cursor: pointer;
    font-size: 12px;
    transition: all 0.3s ease;
}

.mode-btn:hover, .mode-btn.active {
    border-color: #4c63d2;
    background: #4c63d2;
```

```
    color: white;
}

.compose-actions {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-top: 15px;
}

.compose-features {
    display: flex;
    gap: 15px;
}

.feature-btn {
    background: none;
    border: none;
    cursor: pointer;
    padding: 8px;
    border-radius: 6px;
    transition: background 0.3s ease;
}

.feature-btn:hover {
    background: rgba(76, 99, 210, 0.1);
}

.post-btn {
    background: linear-gradient(45deg, #4c63d2, #6c7ce7);
    color: white;
    border: none;
    border-radius: 20px;
    padding: 10px 25px;
    cursor: pointer;
    font-weight: 600;
    transition: all 0.3s ease;
}

.post-btn:hover {
    transform: translateY(-1px);
    box-shadow: 0 4px 15px rgba(76, 99, 210, 0.3);
}

.post-btn:disabled {
    opacity: 0.6;
    cursor: not-allowed;
    transform: none;
```

```
}

.post {
  border-bottom: 1px solid #e0e0e0;
  padding: 20px 0;
}

.post:last-child {
  border-bottom: none;
}

.post-header {
  display: flex;
  align-items: center;
  margin-bottom: 12px;
}

.post-avatar {
  width: 45px;
  height: 45px;
  border-radius: 50%;
  background: linear-gradient(45deg, #667eea, #764ba2);
  display: flex;
  align-items: center;
  justify-content: center;
  color: white;
  font-weight: bold;
  margin-right: 12px;
}

.post-info {
  flex-grow: 1;
}

.post-author {
  font-weight: bold;
  color: #333;
}

.post-handle {
  color: #666;
  font-size: 14px;
}

.post-time {
  color: #666;
  font-size: 14px;
  margin-left: auto;
```

```
}

.post-content {
    margin-bottom: 15px;
    line-height: 1.5;
}

.communication-indicator {
    display: inline-block;
    padding: 2px 8px;
    border-radius: 12px;
    font-size: 11px;
    background: rgba(76, 99, 210, 0.1);
    color: #4c63d2;
    margin-right: 8px;
    margin-bottom: 8px;
}

.post-actions {
    display: flex;
    justify-content: space-around;
    max-width: 400px;
}

.action-btn {
    display: flex;
    align-items: center;
    gap: 8px;
    background: none;
    border: none;
    cursor: pointer;
    padding: 8px 12px;
    border-radius: 20px;
    transition: all 0.3s ease;
    color: #666;
}

.action-btn:hover {
    background: rgba(76, 99, 210, 0.1);
    color: #4c63d2;
}

.trending h3 {
    margin-bottom: 20px;
    color: #4c63d2;
}

.trending-item {
```

```
    padding: 12px 0;
    border-bottom: 1px solid #e0e0e0;
    cursor: pointer;
}

.trending-item:hover {
    color: #4c63d2;
}

.trending-topic {
    font-weight: bold;
    margin-bottom: 5px;
}

.trending-posts {
    font-size: 12px;
    color: #666;
}

.modal {
    display: none;
    position: fixed;
    z-index: 1000;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, 0.5);
}

.modal-content {
    background: white;
    margin: 10% auto;
    padding: 30px;
    border-radius: 15px;
    width: 90%;
    max-width: 500px;
    position: relative;
}

.close-modal {
    position: absolute;
    right: 20px;
    top: 20px;
    font-size: 24px;
    cursor: pointer;
    color: #666;
}
```

```
        .form-group {
            margin-bottom: 20px;
        }

        .form-group label {
            display: block;
            margin-bottom: 5px;
            font-weight: 600;
            color: #4c63d2;
        }

        .form-group input, .form-group textarea, .form-group select {
            width: 100%;
            padding: 12px;
            border: 2px solid #e0e0e0;
            border-radius: 8px;
            font-size: 14px;
            transition: border-color 0.3s ease;
        }

        .form-group input:focus, .form-group textarea:focus, .form-group
select:focus {
            outline: none;
            border-color: #4c63d2;
        }

        .btn {
            background: linear-gradient(45deg, #4c63d2, #6c7ce7);
            color: white;
            border: none;
            border-radius: 8px;
            padding: 12px 24px;
            cursor: pointer;
            font-weight: 600;
            transition: all 0.3s ease;
        }

        .btn:hover {
            transform: translateY(-1px);
            box-shadow: 0 4px 15px rgba(76, 99, 210, 0.3);
        }

        .notification {
            position: fixed;
            top: 20px;
            right: 20px;
            background: #4caf50;
```

```
        color: white;
        padding: 15px 20px;
        border-radius: 8px;
        display: none;
        z-index: 1001;
      }
    </style>


    <div class="container">
      <!-- Header -->
      <div class="header">
        <h1>NeuroConnect</h1>
        <p>Multi-Modal Communication Social Network</p>
        <!-- ðŸ”  AANIC Relay Panel -->
<section class="aanic-relay">
  <h3>ðŸ§  AANIC Relay Console</h3>
  <input type="text" id="aanic-relay-input" placeholder="Enter message">
  <button onclick="transmitAANIC(&#39;station-01&#39;)">Transmit</
button>
  <pre id="aanic-relay-output"></pre>
</section>

<!-- ðŸ§¬ Brain Communication Converter -->
<section class="brain-converter">
  <h3>ðŸ§¬ Brain Communication Codec</h3>
  <input type="text" id="brain-converter-input" placeholder="Enter
word">
  <button
onclick="convertBrainComm(&#39;station-01&#39;)">Convert</button>
  <pre id="brain-converter-output"></pre>
</section>

<!-- ðŸ§  Neural Resonance Visualizer -->
<section class="neural-visualizer">
  <h3>ðŸ§  Neural Resonance Feedback</h3>
  <button onclick="simulateNeuralResonance()">Simulate Resonance</
button>
  <pre id="neural-resonance-output"></pre>
</section>

<!-- ðŸ§  Symbolic Overlay Grid -->
<section class="symbolic-overlay">
  <h3>ðŸ§  Body Alphabet Mapping</h3>
  <input type="text" id="symbolic-input" placeholder="Enter word">
  <button onclick="mapBodyAlphabet()">Map</button>
  <pre id="symbolic-output"></pre>
</section>
```

```
    </div>

    <div class="main-content">
      <!-- Sidebar -->
      <div class="sidebar">
        <div class="profile-section">
          <div class="profile-avatar" onclick="openProfileModal()">
            <span id="profileInitial">U</span>
          </div>
          <div class="profile-info">
            <h3 id="profileName">User</h3>
            <div class="profile-bio" id="profileBio">Digital
communication enthusiast</div>
            <div id="profileIdentifier" style="font-size: 12px; color:
#999;">ID: USER001</div>
          </div>
          <div class="profile-stats">
            <div class="stat">
              <div class="stat-number" id="postCount">0</div>
              <div class="stat-label">Posts</div>
            </div>
            <div class="stat">
              <div class="stat-number" id="followerCount">42</div>
              <div class="stat-label">Connections</div>
            </div>
            <div class="stat">
              <div class="stat-number" id="commModes">4</div>
              <div class="stat-label">Modes</div>
            </div>
          </div> &lt;******Use NeuroConnect to send us an instant
message*******

1. USE AANIC RELAY CONSOLE

Type your message then copy the message first before pressing the
Transmit-button.
NOW PRESS TRANSMIT
2. Now Use the BRAIN COMMUNICATION CODEC;paste the same
message.
NOW PRESS CONVERT
3. CLICK SIMULATE RESONANCE
4. Now use the BODY ALPHABET MAPPING; paste the same message.
NOW CLICK MAP-button/&gt;
          </div>

          <div class="navigation">
            <div class="nav-item"
onclick="showSection(&#39;feed&#39;)">
```

```
                <div class="nav-icon">ðŸ  </div>
                <span>Home Feed</span>
            </div>
            <div class="nav-item"
onclick="showSection(&#39;explore&#39;)">
                <div class="nav-icon">ðŸ”  </div>
                <span>Explore</span>
            </div>
            <div class="nav-item"
onclick="showSection(&#39;messages&#39;)">
                <div class="nav-icon">ðŸ'¬</div>
                <span>Messages</span>
            </div>
            <div class="nav-item"
onclick="showSection(&#39;neural&#39;)">
                <div class="nav-icon">ðŸ§ </div>
                <span>Neural Hub</span>
            </div>
            <div class="nav-item"
onclick="showSection(&#39;settings&#39;)">
                <div class="nav-icon">âš™ï¸ </div>
                <span>Settings</span>
            </div>
        </div>
    </div>

    <!-- Feed -->
    <div class="feed">
        <div class="compose-area">
            <textarea class="compose-input" id="composeInput"
placeholder="What&#39;s on your mind? Express through multiple
communication modes..."></textarea>

            <div class="communication-modes">
                <div class="mode-btn active" data-mode="text">ðŸ"
Text</div>
                <div class="mode-btn" data-mode="visual">ðŸ'ï¸  Visual</
div>
                <div class="mode-btn" data-mode="emotional">â¤ï¸
Emotional</div>
                <div class="mode-btn" data-mode="conceptual">ðŸ'
Conceptual</div>
                <div class="mode-btn" data-mode="symbolic">ðŸ"¤
Symbolic</div>
            </div>

            <div class="compose-actions">
                <div class="compose-features">
```

```
                    <button class="feature-btn" title="Add Media">ðŸ“· </
button>
                    <button class="feature-btn" title="Add Location">ðŸ“ </
button>
                    <button class="feature-btn" title="Schedule">â° </
button>
                    <button class="feature-btn" title="Multi-Mode">ðŸ”„ </
button>
                </div>
                <button class="post-btn" onclick="createPost()">Share</
button>
            </div>
        </div>

        <div id="postsContainer"><div class="post">
            <div class="post-header">
                <div class="post-avatar">A</div>
                <div class="post-info">
                    <span class="post-author">Alex Chen</span>
                    <span class="post-handle">@alex_tech</span>
                </div>
                <span class="post-time">2h</span>
            </div>
            <div class="post-content">
                <div style="margin-bottom: 10px;"><span
class="communication-indicator">ðŸ“ text</span><span
class="communication-indicator">ðŸ’ conceptual</span></div>
                Exploring new ways to express ideas through multi-modal
communication. The future of human connection is fascinating!
            </div>
            <div class="post-actions">
                <button class="action-btn"
onclick="neuroConnect.interactWithPost(1, &#39;reply&#39;)">
                    ðŸ’¬ <span>15</span>
                </button>
                <button class="action-btn"
onclick="neuroConnect.interactWithPost(1, &#39;repost&#39;)">
                    ðŸ”„ <span>8</span>
                </button>
                <button class="action-btn"
onclick="neuroConnect.interactWithPost(1, &#39;like&#39;)">
                    â¤ï¸ <span>42</span>
                </button>
                <button class="action-btn"
onclick="neuroConnect.interactWithPost(1, &#39;share&#39;)">
                    ðŸ“¤
                </button>
            </div>
```

```
            </div><div class="post">
                <div class="post-header">
                    <div class="post-avatar">M</div>
                    <div class="post-info">
                        <span class="post-author">Maya Rivera</span>
                        <span class="post-handle">@maya_creative</span>
                    </div>
                    <span class="post-time">4h</span>
                </div>
                <div class="post-content">
                    <div style="margin-bottom: 10px;"><span
class="communication-indicator">ðŸ“  text</span><span
class="communication-indicator">ðŸ‘ï¸  visual</span><span
class="communication-indicator">â¤ï¸  emotional</span></div>
                    Sometimes words aren't enough. That's why I love
combining visual and emotional expression in my posts.
                </div>
                <div class="post-actions">
                    <button class="action-btn"
onclick="neuroConnect.interactWithPost(2, &#39;reply&#39;)">
                        ðŸ’¬ <span>31</span>
                    </button>
                    <button class="action-btn"
onclick="neuroConnect.interactWithPost(2, &#39;repost&#39;)">
                        ðŸ”„ <span>23</span>
                    </button>
                    <button class="action-btn"
onclick="neuroConnect.interactWithPost(2, &#39;like&#39;)">
                        â¤ï¸  <span>128</span>
                    </button>
                    <button class="action-btn"
onclick="neuroConnect.interactWithPost(2, &#39;share&#39;)">
                        ðŸ“¤
                    </button>
                </div>
            </div><div class="post">
                <div class="post-header">
                    <div class="post-avatar">S</div>
                    <div class="post-info">
                        <span class="post-author">Dr. Sarah Kim</span>
                        <span class="post-handle">@dr_sarah_research</span>
                    </div>
                    <span class="post-time">6h</span>
                </div>
                <div class="post-content">
                    <div style="margin-bottom: 10px;"><span
class="communication-indicator">ðŸ“  text</span><span
class="communication-indicator">ðŸ’ conceptual</span></div>
```

Research shows that multi-modal communication can increase understanding and empathy. Excited to see how this platform develops.

```
            </div>
            <div class="post-actions">
              <button class="action-btn"
onclick="neuroConnect.interactWithPost(3, &#39;reply&#39;)">
                ðŸ’¬ <span>22</span>
              </button>
              <button class="action-btn"
onclick="neuroConnect.interactWithPost(3, &#39;repost&#39;)">
                ðŸ”„ <span>45</span>
              </button>
              <button class="action-btn"
onclick="neuroConnect.interactWithPost(3, &#39;like&#39;)">
                â¤ï¸ <span>87</span>
              </button>
              <button class="action-btn"
onclick="neuroConnect.interactWithPost(3, &#39;share&#39;)">
                ðŸ“¤
              </button>
            </div>
          </div></div>
        </div>

        <!-- Trending Sidebar -->
        <div class="trending">
          <h3>Trending Topics</h3>
          <div class="trending-item">
            <div class="trending-topic">#MultiModalComm</div>
            <div class="trending-posts">1,234 posts</div>
          </div>
          <div class="trending-item">
            <div class="trending-topic">#DigitalEmpathy</div>
            <div class="trending-posts">856 posts</div>
          </div>
          <div class="trending-item">
            <div class="trending-topic">#FutureCommunication</div>
            <div class="trending-posts">642 posts</div>
          </div>
          <div class="trending-item">
            <div class="trending-topic">#TechWellness</div>
            <div class="trending-posts">428 posts</div>
          </div>

          <h3 style="margin-top: 30px;">Communication Modes</h3>
          <div style="font-size: 12px; color: #666; line-height: 1.6;">
            <p><strong>Text:</strong> Traditional written
```

```
communication</p>
                <p><strong>Visual:</strong> Image and symbol-based
expression</p>
                <p><strong>Emotional:</strong> Sentiment and feeling
indicators</p>
                <p><strong>Conceptual:</strong> Abstract idea
representation</p>
                <p><strong>Symbolic:</strong> Pattern and metaphor
communication</p>
            </div>
        </div>
    </div>
</div>

<!-- Profile Modal -->
<div id="profileModal" class="modal">
    <div class="modal-content">
        <span class="close-modal" onclick="closeProfileModal()">Ã—</
span>
        <h2>Edit Profile</h2>
        <div class="form-group">
            <label>Display Name</label>
            <input type="text" id="editName" placeholder="Enter your
name">
        </div>
        <div class="form-group">
            <label>Bio</label>
            <textarea id="editBio" rows="3" placeholder="Tell others about
yourself..."></textarea>
        </div>
        <div class="form-group">
            <label>Profile Identifier</label>
            <select id="editIdentifier">
                <option value="USER">Standard User ID</option>
                <option value="CREATOR">Content Creator ID</option>
                <option value="RESEARCHER">Research Professional</
option>
                <option value="ARTIST">Digital Artist</option>
                <option value="TECH">Technology Enthusiast</option>
            </select>
        </div>
        <div class="form-group">
            <label>Communication Preferences</label>
            <div style="display: flex; gap: 10px; flex-wrap: wrap;">
                <label style="display: flex; align-items: center; gap: 5px;">
                    <input type="checkbox" id="prefText" checked=""> Text
                </label>
                <label style="display: flex; align-items: center; gap: 5px;">
```

```
                <input type="checkbox" id="prefVisual"> Visual
            </label>
            <label style="display: flex; align-items: center; gap: 5px;">
                <input type="checkbox" id="prefEmotional"> Emotional
            </label>
            <label style="display: flex; align-items: center; gap: 5px;">
                <input type="checkbox" id="prefConceptual"> Conceptual
            </label>
        </div>
    </div>
    <button class="btn" onclick="saveProfile()">Save Profile</button>
  </div>
</div>

<!-- Notification -->
<div id="notification" class="notification"></div>

<script>
  class NeuroConnect {
    constructor() {
      this.posts = [];
      this.activeModes = new Set(['text']);
      this.userProfile = {
        name: 'User',
        bio: 'Digital communication enthusiast',
        identifier: 'USER001',
        avatar: 'U'
      };
      this.initializeSamplePosts();
      this.bindEvents();
      this.renderPosts();
    }

    initializeSamplePosts() {
      const samplePosts = [
        {
          id: 1,
          author: 'Alex Chen',
          handle: '@alex_tech',
          time: '2h',
          content: 'Exploring new ways to express ideas through
multi-modal communication. The future of human connection is
fascinating!',
          modes: ['text', 'conceptual'],
          likes: 42,
          reposts: 8,
          replies: 15,
          avatar: 'A'
```

```
            },
            {
                id: 2,
                author: 'Maya Rivera',
                handle: '@maya_creative',
                time: '4h',
                content: 'Sometimes words aren\'t enough. That\'s why I
love combining visual and emotional expression in my posts.',
                modes: ['text', 'visual', 'emotional'],
                likes: 128,
                reposts: 23,
                replies: 31,
                avatar: 'M'
            },
            {
                id: 3,
                author: 'Dr. Sarah Kim',
                handle: '@dr_sarah_research',
                time: '6h',
                content: 'Research shows that multi-modal communication
can increase understanding and empathy. Excited to see how this
platform develops.',
                modes: ['text', 'conceptual'],
                likes: 87,
                reposts: 45,
                replies: 22,
                avatar: 'S'
            }
        ];

        this.posts = samplePosts;
    }

    bindEvents() {
        // Communication mode selection
        document.querySelectorAll('.mode-btn').forEach(btn => {
            btn.addEventListener('click', () => {
                btn.classList.toggle('active');
                const mode = btn.dataset.mode;

                if (this.activeModes.has(mode)) {
                    this.activeModes.delete(mode);
                } else {
                    this.activeModes.add(mode);
                }
            });
        });
    }
```

```
        createPost() {
            const content =
document.getElementById('composeInput').value.trim();
            if (!content) {
                this.showNotification('Please enter some content', 'error');
                return;
            }

            const newPost = {
                id: this.posts.length + 1,
                author: this.userProfile.name,
                handle: '@' + this.userProfile.name.toLowerCase().replace(' ', '_'),
                time: 'now',
                content: content,
                modes: Array.from(this.activeModes),
                likes: 0,
                reposts: 0,
                replies: 0,
                avatar: this.userProfile.avatar
            };

            this.posts.unshift(newPost);
            document.getElementById('composeInput').value = '';
            this.renderPosts();
            this.updatePostCount();
            this.showNotification('Post shared successfully!');
        }

        renderPosts() {
            const container = document.getElementById('postsContainer');
            container.innerHTML = '';

            this.posts.forEach(post => {
                const postElement = this.createPostElement(post);
                container.appendChild(postElement);
            });
        }

        createPostElement(post) {
            const postDiv = document.createElement('div');
            postDiv.className = 'post';

            const modesHtml = post.modes.map(mode =>
                `<span class="communication-indicator">${
{this.getModeIcon(mode)} ${mode}</span>`
            ).join('');
```

```
            postDiv.innerHTML = `
                <div class="post-header">
                    <div class="post-avatar">${post.avatar}</div>
                    <div class="post-info">
                        <span class="post-author">${post.author}</span>
                        <span class="post-handle">${post.handle}</span>
                    </div>
                    <span class="post-time">${post.time}</span>
                </div>
                <div class="post-content">
                    <div style="margin-bottom: 10px;">${modesHtml}</div>
                    ${post.content}
                </div>
                <div class="post-actions">
                    <button class="action-btn"
onclick="neuroConnect.interactWithPost(${post.id}, 'reply')">
                        ðŸ’¬ <span>${post.replies}</span>
                    </button>
                    <button class="action-btn"
onclick="neuroConnect.interactWithPost(${post.id}, 'repost')">
                        ðŸ”„ <span>${post.reposts}</span>
                    </button>
                    <button class="action-btn"
onclick="neuroConnect.interactWithPost(${post.id}, 'like')">
                        â¤ï¸ <span>${post.likes}</span>
                    </button>
                    <button class="action-btn"
onclick="neuroConnect.interactWithPost(${post.id}, 'share')">
                        ðŸ“¤
                    </button>
                </div>
            `;

            return postDiv;
        }

        getModeIcon(mode) {
            const icons = {
                text: 'ðŸ“ ',
                visual: 'ðŸ‘ï¸ ',
                emotional: 'â¤ï¸ ',
                conceptual: 'ðŸ§ ',
                symbolic: 'ðŸ”¤'
            };
            return icons[mode] || 'ðŸ“ ';
        }

        interactWithPost(postId, action) {
```

```
            const post = this.posts.find(p => p.id === postId);
            if (!post) return;

            switch(action) {
                case 'like':
                    post.likes++;
                    this.showNotification('Post liked!');
                    break;
                case 'repost':
                    post.reposts++;
                    this.showNotification('Post reposted!');
                    break;
                case 'reply':
                    this.showNotification('Reply feature coming soon!');
                    break;
                case 'share':
                    this.showNotification('Post shared!');
                    break;
            }

            this.renderPosts();
        }

        openProfileModal() {
            document.getElementById('editName').value =
this.userProfile.name;
            document.getElementById('editBio').value = this.userProfile.bio;
            document.getElementById('profileModal').style.display = 'block';
        }

        closeProfileModal() {
            document.getElementById('profileModal').style.display = 'none';
        }

        saveProfile() {
            const name =
document.getElementById('editName').value.trim();
            const bio = document.getElementById('editBio').value.trim();
            const identifier =
document.getElementById('editIdentifier').value;

            if (name) {
                this.userProfile.name = name;
                this.userProfile.bio = bio || 'Digital communication enthusiast';
                this.userProfile.identifier = identifier +
String(Math.floor(Math.random() * 1000)).padStart(3, '0');
                this.userProfile.avatar = name.charAt(0).toUpperCase();
```

```
            this.updateProfileDisplay();
            this.closeProfileModal();
            this.showNotification('Profile updated successfully!');
        }
    }

    updateProfileDisplay() {
        document.getElementById('profileName').textContent =
this.userProfile.name;
        document.getElementById('profileBio').textContent =
this.userProfile.bio;
        document.getElementById('profileIdentifier').textContent = 'ID: '
+ this.userProfile.identifier;
        document.getElementById('profileInitial').textContent =
this.userProfile.avatar;
    }

    updatePostCount() {
        const userPosts = this.posts.filter(post => post.author ===
this.userProfile.name);
        document.getElementById('postCount').textContent =
userPosts.length;
    }

    showNotification(message, type = 'success') {
        const notification = document.getElementById('notification');
        notification.textContent = message;
        notification.style.display = 'block';
        notification.style.background = type === 'error' ? '#f44336' :
'#4caf50';

        setTimeout(() => {
            notification.style.display = 'none';
        }, 3000);
    }

    showSection(section) {
        // Placeholder for navigation - would expand in full
implementation
        this.showNotification(`${section} section - Feature in
development`);
    }
}

// Global functions for onclick handlers
let neuroConnect;

function openProfileModal() {
```

```
        neuroConnect.openProfileModal();
    }

    function closeProfileModal() {
        neuroConnect.closeProfileModal();
    }

    function saveProfile() {
        neuroConnect.saveProfile();
    }

    function createPost() {
        neuroConnect.createPost();
    }

    function showSection(section) {
        neuroConnect.showSection(section);
    }

    // Initialize the application
    document.addEventListener('DOMContentLoaded', function() {
        neuroConnect = new NeuroConnect();
    });

    // Close modal when clicking outside
    window.onclick = function(event) {
        const modal = document.getElementById('profileModal');
        if (event.target == modal) {
            modal.style.display = 'none';
        }
    }
</script>


s



<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<title>CNB Wearable Brain Reader System</title>
<style>
    * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
    }
```

```
        body {
            font-family: 'Courier New', monospace;
            background: linear-gradient(135deg, #1a1a2e 0%, #16213e 50%,
#0f3460 100%);
            color: #00ff88;
            min-height: 100vh;
            overflow-x: auto;
        }

        .container {
            max-width: 1400px;
            margin: 0 auto;
            padding: 15px;
        }

        .header {
            text-align: center;
            margin-bottom: 20px;
            padding: 20px;
            background: rgba(0, 255, 136, 0.1);
            border: 2px solid #00ff88;
            border-radius: 10px;
        }

        .header h1 {
            font-size: 24px;
            margin-bottom: 10px;
            text-shadow: 0 0 10px #00ff88;
        }

        .warning {
            background: #ff4444;
            color: white;
            padding: 15px;
            border-radius: 8px;
            margin-bottom: 20px;
            font-size: 14px;
            font-weight: bold;
        }

        .main-grid {
            display: grid;
            grid-template-columns: 1fr 1fr;
            gap: 20px;
            margin-bottom: 20px;
        }

        @media (max-width: 768px) {
```

```
        .main-grid {
            grid-template-columns: 1fr;
        }
    }

    .panel {
        background: rgba(0, 40, 80, 0.8);
        border: 1px solid #00ff88;
        border-radius: 8px;
        padding: 15px;
    }

    .panel h2 {
        color: #00ff88;
        margin-bottom: 15px;
        font-size: 18px;
        text-transform: uppercase;
    }

    .cnb-visualizer {
        display: flex;
        justify-content: center;
        align-items: center;
        height: 200px;
        border: 2px dashed #00ff88;
        margin: 15px 0;
        position: relative;
        background: radial-gradient(circle at center, rgba(0, 255, 136, 0.1),
transparent);
    }

    .cnb-bridge {
        width: 120px;
        height: 80px;
        border: 3px solid #00ff88;
        border-radius: 10px;
        position: relative;
        animation: pulse 2s infinite;
        display: flex;
        align-items: center;
        justify-content: center;
        font-weight: bold;
        font-size: 12px;
    }

    .nerve-line {
        position: absolute;
        width: 2px;
```

```
    height: 60px;
    background: linear-gradient(to bottom, #00ff88, transparent);
    animation: nerve-signal 1.5s infinite;
}

.nerve-line:nth-child(2) { left: 20px; animation-delay: 0.2s; }
.nerve-line:nth-child(3) { right: 20px; animation-delay: 0.4s; }
.nerve-line:nth-child(4) { left: 60px; animation-delay: 0.6s; }

@keyframes pulse {
    0%, 100% { opacity: 0.7; transform: scale(1); }
    50% { opacity: 1; transform: scale(1.05); }
}

@keyframes nerve-signal {
    0% { height: 0; }
    50% { height: 60px; }
    100% { height: 0; }
}

.codec-input {
    width: 100%;
    background: #000;
    color: #00ff88;
    border: 1px solid #00ff88;
    padding: 8px;
    font-family: 'Courier New', monospace;
    font-size: 12px;
    margin-bottom: 10px;
}

.codec-btn {
    background: linear-gradient(45deg, #00ff88, #00cc66);
    color: #000;
    border: none;
    padding: 8px 15px;
    font-weight: bold;
    cursor: pointer;
    margin: 5px;
    font-family: 'Courier New', monospace;
}

.codec-btn:hover {
    background: linear-gradient(45deg, #00cc66, #00aa44);
}

.terminal {
    background: #000;
```

```
    color: #00ff88;
    padding: 10px;
    font-size: 11px;
    height: 200px;
    overflow-y: auto;
    border: 1px solid #00ff88;
    margin-bottom: 10px;
}

.wearable-books {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
    gap: 10px;
    margin-bottom: 15px;
}

.book-item {
    background: linear-gradient(135deg, #003366, #004488);
    border: 2px solid #00ff88;
    border-radius: 8px;
    padding: 12px;
    cursor: pointer;
    transition: all 0.3s ease;
    text-align: center;
}

.book-item:hover {
    transform: translateY(-3px);
    box-shadow: 0 5px 15px rgba(0, 255, 136, 0.3);
}

.book-item.active {
    background: linear-gradient(135deg, #00ff88, #00cc66);
    color: #000;
}

.conversion-display {
    background: #001122;
    border: 1px solid #00ff88;
    padding: 15px;
    margin: 10px 0;
    border-radius: 5px;
}

.brain-commands {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(120px, 1fr));
    gap: 8px;
```

```
            margin-bottom: 15px;
        }

        .brain-cmd {
            background: #004466;
            border: 1px solid #00ff88;
            color: #00ff88;
            padding: 8px;
            font-size: 11px;
            cursor: pointer;
            text-align: center;
        }

        .brain-cmd:hover {
            background: #00ff88;
            color: #000;
        }

        .algorithm-display {
            background: #000;
            color: #00ff88;
            padding: 15px;
            font-size: 10px;
            height: 300px;
            overflow-y: auto;
            border: 1px solid #00ff88;
        }
    </style>


    <div class="container">
        <div class="header">
            <h1>CNB WEARABLE BRAIN READER SYSTEM</h1>
            <p>Central Nerve Bridge Interface with Wearable Brain Books</p>
            <p style="font-size: 14px; opacity: 0.8;">Theoretical Conversion
Codec Implementation</p>
        </div>

        <div class="warning">
            âš ï¸ It works but use with care and do your own research this is
the disclaimer. (davidgomadza liveforever@buythecurefordeath.world)
        </div>

        <div class="main-grid">
            <div class="panel">
                <h2>Central Nerve Bridge Visualizer</h2>
                <div class="cnb-visualizer">
                    <div class="cnb-bridge">
```

```
                CNB ACTIVE
                <div class="nerve-line"></div>
                <div class="nerve-line"></div>
                <div class="nerve-line"></div>
                <div class="nerve-line"></div>
            </div>
        </div>
        <div>
            <input type="text" class="codec-input" id="cnbInput"
placeholder="Enter thought pattern for CNB processing...">
            <button class="codec-btn"
onclick="processCNBSignal()">PROCESS CNB</button>
            <button class="codec-btn"
onclick="convertToCodec()">CONVERT CODEC</button>
        </div>
    </div>

    <div class="panel">
        <h2>Wearable Brain Books Library</h2>
        <div class="wearable-books">
            <div class="book-item" onclick="selectBook(&#39;thoughts-
to-audio&#39;, this)">
                <div>Thoughts to Audio</div>
                <div style="font-size: 10px;">Code: 76854</div>
            </div>
            <div class="book-item" onclick="selectBook(&#39;brain-
decoder&#39;, this)">
                <div>Brain Decoder</div>
                <div style="font-size: 10px;">Code: 789838</div>
            </div>
            <div class="book-item" onclick="selectBook(&#39;neural-
interface&#39;, this)">
                <div>Neural Interface</div>
                <div style="font-size: 10px;">Code: 867838692</div>
            </div>
            <div class="book-item" onclick="selectBook(&#39;cnb-
protocols&#39;, this)">
                <div>CNB Protocols</div>
                <div style="font-size: 10px;">Code: 888800</div>
            </div>
        </div>
        <div>
            <button class="codec-btn"
onclick="uploadBrainBook()">UPLOAD TO CNB</button>
            <button class="codec-btn"
onclick="embedNeuralData()">EMBED NEURAL</button>
        </div>
    </div>
```

```
    </div>

    <div class="panel">
      <h2>CNB Signal Processing Terminal</h2>
      <div class="terminal" id="cnbTerminal">[14:49:03] CNB Wearable
Brain Reader System initialized<br>[14:49:03] WARNING: Operating in
theoretical mode only<br></div>
    </div>

    <div class="main-grid">
      <div class="panel">
        <h2>Brain Command Converter</h2>
        <div class="brain-commands">
          <div class="brain-cmd"
onclick="executeCommand(&#39;read-thoughts&#39;)">READ
THOUGHTS</div>
          <div class="brain-cmd"
onclick="executeCommand(&#39;decode-emotions&#39;)">DECODE
EMOTIONS</div>
          <div class="brain-cmd"
onclick="executeCommand(&#39;extract-memories&#39;)">EXTRACT
MEMORY</div>
          <div class="brain-cmd"
onclick="executeCommand(&#39;upload-knowledge&#39;)">UPLOAD
KNOWLEDGE</div>
          <div class="brain-cmd"
onclick="executeCommand(&#39;neural-sync&#39;)">NEURAL SYNC</
div>
          <div class="brain-cmd"
onclick="executeCommand(&#39;thought-stream&#39;)">THOUGHT
STREAM</div>
        </div>
        <div class="conversion-display" id="conversionDisplay">
          Conversion Result: Awaiting CNB signal processing...
        </div>
      </div>

      <div class="panel">
        <h2>CNB Conversion Algorithms</h2>
        <div class="algorithm-display" id="algorithmDisplay">
// CNB THEORETICAL CONVERSION CODEC<br>
// WARNING: Based on unverified anatomical claims<br><br>
class CNBConverter {<br>
  constructor() {<br>
    this.nerveBridge = null; // Theoretical
construct<br>
    this.signalBuffer = [];<br>
    this.conversionMatrix =
```

```
this.initMatrix();<br>
  }<br><br>
  initMatrix() {<br>
    // Hypothetical nerve convergence mapping<br>
    return {<br>
      'thought': [0.2, 0.8, 0.6, 0.3],<br>
      'emotion': [0.7, 0.1, 0.9, 0.4],<br>
      'memory': [0.3, 0.6, 0.2, 0.8]<br>
    };<br>
  }<br><br>
  processCNBSignal(input) {<br>
    // Theoretical signal processing<br>
    if (!this.validateCNB()) {<br>
      return "CNB_ERROR: Bridge not found";<br>
    }<br>
    return this.convertToCodec(input);<br>
  }<br><br>
  convertToCodec(signal) {<br>
    // Simulated conversion process<br>
    const encoded = this.encodeNeural(signal);<br>
    return this.formatOutput(encoded);<br>
  }<br>
}<br>

// CNB Signal Validation<br>
validateCNB() {<br>
  // No actual CNB exists in human anatomy<br>
  console.warn("CNB is theoretical construct");<br>
  return false; // Always returns false<br>
}<br><br>
// Theoretical Nerve Signal Processing<br>
encodeNeural(input) {<br>
  const hash = this.simpleHash(input);<br>
  return `NEURAL_${hash}_THEORETICAL`;<br>
}<br><br>
// Note: This codec cannot function as described<br>
// Real BCIs require direct neural interface
        </div>
        </div>
    </div>

    <div style="text-align: center; margin-top: 20px; padding: 15px;
```

```
background: rgba(0, 255, 136, 0.1); border: 1px solid #00ff88;">
            <p style="color: #ff4444; font-weight: bold;">IMPORTANT: This
interface is purely theoretical. The "Central Nerve Bridge" is not a real
anatomical structure. Current neurotechnology cannot perform the
functions described here.</p>
        </div>
    </div>

    <script>
        let selectedBook = null;
        let cnbLog = [];

        function logToCNB(message) {
            const terminal = document.getElementById('cnbTerminal');
            const timestamp = new Date().toLocaleTimeString();
            cnbLog.push(`[${timestamp}] ${message}`);
            terminal.innerHTML = cnbLog.slice(-15).join('<br>') + '<br>';
            terminal.scrollTop = terminal.scrollHeight;
        }

        function processCNBSignal() {
            const input = document.getElementById('cnbInput').value.trim();
            if (!input) {
                logToCNB('ERROR: No input signal provided');
                return;
            }

            logToCNB(`Processing CNB signal: "${input}"`);
            logToCNB('Searching for nerve convergence patterns...');

            setTimeout(() => {
                logToCNB('WARNING: CNB anatomical structure not located');
                logToCNB('Switching to theoretical processing mode');
                logToCNB(`Signal processed using conceptual algorithms`);
                updateConversionDisplay(`Theoretical conversion of "${input}"
complete`);
            }, 2000);
        }

        function convertToCodec() {
            const input = document.getElementById('cnbInput').value.trim();
            if (!input) {
                logToCNB('ERROR: No signal to convert');
                return;
            }

            logToCNB('Initiating codec conversion...');
            logToCNB('Applying theoretical neural mapping');
```

```
        setTimeout(() => {
            const codec = generateTheoreticalCodec(input);
            logToCNB(`Codec generated: ${codec}`);
            updateConversionDisplay(`Codec: ${codec} | Status:
Theoretical`);
        }, 1500);
    }

    function generateTheoreticalCodec(input) {
        // Generate a plausible-looking but non-functional codec
        const baseCode = input.split('').map(c => c.charCodeAt(0)).join('');
        return `CNB_${baseCode.substring(0, 8)}_$
{Date.now().toString().slice(-6)}`;
    }

    function selectBook(bookId, element) {
        document.querySelectorAll('.book-item').forEach(item => {
            item.classList.remove('active');
        });
        element.classList.add('active');
        selectedBook = bookId;
        logToCNB(`Selected wearable brain book: ${bookId}`);
    }

    function uploadBrainBook() {
        if (!selectedBook) {
            logToCNB('ERROR: No brain book selected');
            return;
        }

        logToCNB(`Uploading brain book: ${selectedBook} to CNB
interface`);
        logToCNB('Establishing neural pathway connections...');

        setTimeout(() => {
            logToCNB('Upload complete - Data integrated into CNB matrix');
            logToCNB('Brain book accessible via thought interface');
        }, 2000);
    }

    function embedNeuralData() {
        if (!selectedBook) {
            logToCNB('ERROR: No brain book selected for embedding');
            return;
        }

        logToCNB('Embedding neural data patterns...');
```

```
        logToCNB('Converting book content to nerve impulse format...');

        setTimeout(() => {
            logToCNB('Neural embedding complete');
            logToCNB('Data now accessible at thought-speed');
        }, 1800);
    }

    function executeCommand(command) {
        logToCNB(`Executing brain command: ${command}`);

        const responses = {
            'read-thoughts': 'Scanning for thought patterns... No CNB
detected',
            'decode-emotions': 'Emotional pattern analysis initiated',
            'extract-memories': 'Memory extraction requires valid CNB
connection',
            'upload-knowledge': 'Knowledge upload to theoretical CNB
initiated',
            'neural-sync': 'Attempting neural synchronization...',
            'thought-stream': 'Thought stream monitoring activated'
        };

        setTimeout(() => {
            logToCNB(responses[command] || 'Command processed');
            updateConversionDisplay(`Command: ${command} | Status:
Simulated`);
        }, 1000);
    }

    function updateConversionDisplay(message) {
        document.getElementById('conversionDisplay').textContent =
`Conversion Result: ${message}`;
    }

    function updateAlgorithmDisplay() {
        const display = document.getElementById('algorithmDisplay');
        const newCode = `
// CNB Signal Validation<br>
validateCNB() {<br>
  // No actual CNB exists in human anatomy<br>
  console.warn("CNB is theoretical construct");<br>
  return false; // Always returns false<br>
}<br><br>
// Theoretical Nerve Signal Processing<br>
encodeNeural(input) {<br>
  const hash = this.simpleHash(input);<br>
  return \`NEURAL_\${hash}_THEORETICAL\`;<br>
```

```
}< br>< br>
// Note: This codec cannot function as described< br>
// Real BCIs require direct neural interface
          `;
          display.innerHTML += newCode;
      }

      // Initialize the interface
      document.addEventListener('DOMContentLoaded', () => {
          logToCNB('CNB Wearable Brain Reader System initialized');
          logToCNB('WARNING: Operating in theoretical mode only');

          setTimeout(updateAlgorithmDisplay, 3000);

          // Simulate periodic CNB scanning
          setInterval(() => {
              if (Math.random() < 0.1) {
                  const activities = [
                      'CNB scan: No anatomical structure found',
                      'Theoretical nerve mapping updated',
                      'Brain book interface ready',
                      'Conversion codec standing by'
                  ];
                  logToCNB(activities[Math.floor(Math.random() *
activities.length)]);
              }
          }, 8000);
      });
  </script>




  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Non-Electric Brain Decoder - Prototype Interface</title>
  <style>
      * {
          margin: 0;
          padding: 0;
          box-sizing: border-box;
      }

      body {
          font-family: 'Courier New', monospace;
          background: linear-gradient(135deg, #0f0f23 0%, #1a1a3e 50%,
#2d1b69 100%);
```

```
    color: #00ff41;
    min-height: 100vh;
    overflow-x: auto;
}

.container {
    max-width: 1400px;
    margin: 0 auto;
    padding: 15px;
}

.header {
    text-align: center;
    margin-bottom: 20px;
    padding: 20px;
    background: rgba(0, 255, 65, 0.1);
    border: 2px solid #00ff41;
    border-radius: 10px;
}

.header h1 {
    font-size: 24px;
    margin-bottom: 10px;
    text-shadow: 0 0 10px #00ff41;
}

.disclaimer {
    background: #ff4444;
    color: white;
    padding: 15px;
    border-radius: 8px;
    margin-bottom: 20px;
    font-size: 14px;
    font-weight: bold;
}

.main-grid {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 20px;
    margin-bottom: 20px;
}

@media (max-width: 768px) {
    .main-grid {
        grid-template-columns: 1fr;
    }
}
```

```
.panel {
    background: rgba(0, 20, 40, 0.8);
    border: 1px solid #00ff41;
    border-radius: 8px;
    padding: 15px;
}

.panel h2 {
    color: #00ff41;
    margin-bottom: 15px;
    font-size: 18px;
    text-transform: uppercase;
}

.create-code-input {
    width: 100%;
    background: #000;
    color: #00ff41;
    border: 1px solid #00ff41;
    padding: 8px;
    font-family: 'Courier New', monospace;
    font-size: 12px;
    margin-bottom: 10px;
}

.execute-btn {
    background: linear-gradient(45deg, #00ff41, #00cc33);
    color: #000;
    border: none;
    padding: 8px 15px;
    font-weight: bold;
    cursor: pointer;
    margin: 5px;
    font-family: 'Courier New', monospace;
}

.execute-btn:hover {
    background: linear-gradient(45deg, #00cc33, #009922);
}

.terminal {
    background: #000;
    color: #00ff41;
    padding: 10px;
    font-size: 11px;
    height: 200px;
    overflow-y: auto;
```

```
    border: 1px solid #00ff41;
    margin-bottom: 10px;
}

.oax-visualizer {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 150px;
    border: 2px dashed #00ff41;
    margin: 15px 0;
    position: relative;
}

.oax-rotary {
    width: 60px;
    height: 60px;
    border: 3px solid #00ff41;
    border-radius: 50%;
    position: relative;
    animation: rotate 3s linear infinite;
}

.oax-rotary::before {
    content: '';
    position: absolute;
    top: -5px;
    left: 50%;
    transform: translateX(-50%);
    width: 6px;
    height: 70px;
    background: #00ff41;
}

@keyframes rotate {
    from { transform: rotate(0deg); }
    to { transform: rotate(360deg); }
}

.brain-status {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
    gap: 10px;
    margin-bottom: 15px;
}

.status-item {
    background: rgba(0, 255, 65, 0.1);
```

```
            border: 1px solid #00ff41;
            padding: 8px;
            text-align: center;
            font-size: 11px;
        }

        .codec-display {
            background: #000;
            color: #00ff41;
            padding: 10px;
            font-size: 10px;
            height: 300px;
            overflow-y: auto;
            border: 1px solid #00ff41;
        }

        .controls-grid {
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(120px, 1fr));
            gap: 8px;
            margin-bottom: 15px;
        }

        .thought-extractor {
            background: rgba(255, 0, 0, 0.2);
            border: 2px solid #ff0000;
            padding: 15px;
            margin: 15px 0;
            border-radius: 8px;
        }

        .thought-extractor h3 {
            color: #ff4444;
            margin-bottom: 10px;
        }
    </style>


    <div class="container">
        <div class="header">
            <h1>WORLD'S FIRST NON-ELECTRIC POWERED BRAIN
DECODER/READER</h1>
            <p>PROTOTYPE INTERFACE - CONCEPTUAL DEMONSTRATION
ONLY</p>
            <p>Database ID: OAXARATEDAVIDGOMADZA-7628983868</p>
        </div>

        <div class="disclaimer">
```

âš ï¸ CRITICAL DISCLAIMER: This is a prototype interface for conceptual demonstration only. The underlying technology claims are not scientifically validated. Current brain-computer interfaces require electrical power, direct neural contact, and extensive calibration. This interface does perform actual brain reading BUT is not tested extensively.

<h2>OAX Rotary Control System</h2>

OAX Position
85Â° North

Rotary Speed
0 RPM

Long Ago Value
8.00 sec

Atererean Level
0.869838xy

<h2>Create Code Execution Terminal</h2>
<textarea class="create-code-input" id="createCodeInput" rows="3" placeholder="Enter CREATE codes here..."></textarea>

<button class="execute-btn" onclick="executeCreateCode()">EXECUTE</button>
<button class="execute-btn" onclick="initializeBrainReader()">INIT READER</button>
<button class="execute-btn" onclick="startDecoding()">START DECODE</button>
<button class="execute-btn" onclick="cloneBrain()">CLONE BRAIN</button>

```
            &gt; System initialized...< br>
            &gt; Awaiting CREATE commands...< br>
            &gt; Non- electric power source: STANDBY< br>
          < /div>
        < /div>
      < /div>

      < div class="thought- extractor">
        < h3> Digital Brain Thoughts Extractor (086795xtu)< /h3>
        < input type="file" accept="image/*" id="imageInput"
onchange="extractThoughts()">
        < button class="execute- btn"
onclick="document.getElementById(&#39;imageInput&#39;).click()"> Loa
d Image< /button>
        < div class="terminal" id="thoughtOutput" style="height: 120px;">
          &gt; Ready to extract thoughts from images...< br>
          &gt; WARNING: This is conceptual demonstration only< br>
        < /div>
      < /div>

      < div class="main- grid">
        < div class="panel">
          < h2> Brain Command Center< /h2>
          < div class="controls- grid">
            < button class="execute- btn"
onclick="brainCommand(&#39;jump&#39;)"> ASK TO JUMP< /button>
            < button class="execute- btn"
onclick="brainCommand(&#39;split&#39;)"> ASK TO SPLIT< /button>
            < button class="execute- btn"
onclick="brainCommand(&#39;merge&#39;)"> ASK TO MERGE< /button>
            < button class="execute- btn"
onclick="brainCommand(&#39;reveal&#39;)"> ASK REVEAL< /button>
            < button class="execute- btn"
onclick="brainCommand(&#39;clone&#39;)"> START CLONING< /button>
            < button class="execute- btn"
onclick="brainCommand(&#39;reset&#39;)"> BRAIN RESET< /button>
          < /div>
          < div class="terminal" id="brainTerminal" style="height: 150px;">
            &gt; Brain command interface ready...< br>
          < /div>
        < /div>

        < div class="panel">
          < h2> Neural Codecs &amp; Transmitter< /h2>
          < div class="codec- display" id="codecDisplay">
// NON- ELECTRIC POWERED BRAIN DECODER CODECS< br>
// Java Implementation for Mobile Integration< br>< br>
public class BrainDecoderCodec {< br>
```

```
  private double atereреanLevel = 0.869838;<br>
  private int longAgoValue = 8; // seconds<br>
  private String oaxPosition = "85Â° North, 28Â° South";<br>
  <br>
  public void initializeDecoder() {<br>
    // WARNING: Conceptual code only<br>
    System.out.println("Decoder
initialized");<br>
  }<br>
  <br>
  public String processCreateCode(String code) {<br>
    if (code.startsWith("create.")) {<br>
      return
executeCommand(code);<br>
    }<br>
    return "Invalid command format";<br>
  }<br>
  <br>
  private String executeCommand(String cmd) {<br>
    // Simulate command processing<br>
    return "Command executed: " + cmd;<br>
  }<br>
}<br><br>
// Coaxial Rotary Propeller Controller<br>
class OAXRotaryController {<br>
  private double rotationAngle = 85.0;<br>
  private int rotationSpeed = 0;<br>
  <br>
  public void activateRotary() {<br>
    rotationSpeed = 120; // RPM<br>
    System.out.println("OAX Rotary
activated");<br>
  }<br>
}<br>
            </div>
        </div>
    </div>

    <div class="panel">
        <h2>Mobile Phone Integration Prototype</h2>
        <p style="color: #00ff41; margin-bottom: 10px;">Proposed
smartphone implementation (CONCEPTUAL ONLY):</p>
        <div class="terminal" id="mobileSpecs" style="height: 180px;">
&gt; HARDWARE REQUIREMENTS (THEORETICAL):<br>
&gt; - Quantum resonance sensor array<br>
&gt; - Non-electric neural field detector<br>
&gt; - Coaxial micro-rotary assembly<br>
&gt; - Thought pattern recognition chip<br>
```

```
&gt; - Neural frequency transmitter/receiver<br><br>
&gt; SOFTWARE INTEGRATION:<br>
&gt; - BrainDecoder.apk (Android)<br>
&gt; - Neural pattern recognition AI<br>
&gt; - CREATE code interpreter<br>
&gt; - Real-time thought visualization<br><br>
&gt; STATUS: PROTOTYPE CONCEPT ONLY<br>
&gt; REALITY CHECK: Current technology limitations prevent actual implementation
        </div>
      </div>

      <div style="text-align: center; margin-top: 20px; padding: 15px; background: rgba(0, 255, 65, 0.1); border: 1px solid #00ff41;">
        <p style="color: #ff4444; font-weight: bold;">IMPORTANT: This interface demonstrates concepts from the provided documents but does not constitute functional brain reading technology. All outputs are simulated for demonstration purposes.</p>
      </div>
    </div>

    <script>
      let systemLog = [];
      let isDecodingActive = false;

      function logToTerminal(terminalId, message) {
        const terminal = document.getElementById(terminalId);
        const timestamp = new Date().toLocaleTimeString();
        terminal.innerHTML += `[${timestamp}] ${message}<br>`;
        terminal.scrollTop = terminal.scrollHeight;
      }

      function executeCreateCode() {
        const code = document.getElementById('createCodeInput').value.trim();
        if (!code) {
          logToTerminal('terminal', 'ERROR: No CREATE code provided');
          return;
        }

        if (code.startsWith('create.')) {
          logToTerminal('terminal', `Executing: ${code}`);
          setTimeout(() => {
            logToTerminal('terminal', 'Command processed - Non-electric systems responding');
            updateSystemStatus();
          }, 1000);
        } else {
```

```
            logToTerminal('terminal', 'ERROR: Invalid CREATE code format');
        }
    }

    function initializeBrainReader() {
        logToTerminal('terminal', 'Initializing brain reader systems...');
        logToTerminal('terminal', 'Loading atererean power source...');
        logToTerminal('terminal', 'Calibrating OAX rotary position...');
        setTimeout(() => {
            logToTerminal('terminal', 'Brain reader initialized - Ready for
decoding');
            updateOAXStatus();
        }, 2000);
    }

    function startDecoding() {
        if (!isDecodingActive) {
            isDecodingActive = true;
            logToTerminal('terminal', 'Starting brain decoding process...');
            logToTerminal('terminal', 'Activating non-electric neural
sensors...');
            logToTerminal('terminal', 'Scanning for thought patterns...');

            setTimeout(() => {
                logToTerminal('terminal', 'SIMULATION: Thought patterns
detected');
                logToTerminal('terminal', 'WARNING: Actual brain reading not
possible');
            }, 1500);
        } else {
            logToTerminal('terminal', 'Decoding already active');
        }
    }

    function cloneBrain() {
        logToTerminal('terminal', 'Initiating brain cloning sequence...');
        logToTerminal('terminal', 'Using adter duplication method...');
        setTimeout(() => {
            logToTerminal('terminal', 'Brain clone created - sending to
magnar storage');
            logToTerminal('terminal', 'Original thoughts preserved');
        }, 2000);
    }

    function brainCommand(command) {
        const commands = {
            jump: 'create.asktojump.start',
            split: 'create.asktosplit.start',
```

```
            merge: 'create.asktomerge.start',
            reveal: 'create.asktoreveal.start',
            clone: 'create.startbraincloningorduplication.start',
            reset: 'create.brainreset086688xtu.start'
        };

        logToTerminal('brainTerminal', `Executing: $
{commands[command]}`);
        setTimeout(() => {
            logToTerminal('brainTerminal', `Brain command '${command}'
processed`);
        }, 800);
    }

    function extractThoughts() {
        const file = document.getElementById('imageInput').files[0];
        if (file) {
            logToTerminal('thoughtOutput', `Processing image: ${file.name}
`);
            logToTerminal('thoughtOutput', 'Analyzing neural patterns in
image...');
            setTimeout(() => {
                logToTerminal('thoughtOutput', 'SIMULATION RESULT:
Thought extraction complete');
                logToTerminal('thoughtOutput', 'ACTUAL CAPABILITY: Cannot
read thoughts from images');
            }, 2000);
        }
    }

    function updateSystemStatus() {
        document.getElementById('rotary-speed').textContent =
Math.floor(Math.random() * 200) + ' RPM';
        document.getElementById('long-ago').textContent = (8.0 +
Math.random() * 0.1).toFixed(2) + ' sec';
    }

    function updateOAXStatus() {
        const angles = ['85Â° North', '72Â° Northeast', '90Â° East', '78Â°
Southeast'];
        document.getElementById('oax-position').textContent =
angles[Math.floor(Math.random() * angles.length)];
    }

    // Simulate periodic system updates
    setInterval(() => {
        if (Math.random() < 0.1) {
            const updates = [
```

```
                'Neural field fluctuation detected',
                'OAX rotary calibration stable',
                'Non-electric power maintaining levels',
                'Thought pattern buffer ready'
            ];
            logToTerminal('terminal', updates[Math.floor(Math.random() *
updates.length)]);
        }
    }, 5000);

    // Initialize system on load
    document.addEventListener('DOMContentLoaded', () => {
        logToTerminal('terminal', 'Non-electric brain decoder prototype
loaded');
        logToTerminal('terminal', 'System ready for demonstration');
    });
  </script>


 s



<meta name="viewport" content="width=device-width,initial-scale=1">
<title>AANIC â€” FFT Spectral EM â†’ Written (by David Gomadza)</
title>
<style>
 :root{--bg:#071017;--panel:#0f1720;--accent:#00ffd5;--muted:#9fb;}
 body{background:var(--bg);color:#e6fff9;font-family:Inter,Segoe
UI,Arial;padding:20px}
 h1{color:var(--accent);margin:0 0 6px}
 .brand{font-size:13px;color:var(--muted);margin-bottom:12px}
 .grid{display:grid;grid-template-columns:1fr 420px;gap:18px}
 .panel{background:var(--panel);padding:14px;border-radius:10px;box-
shadow:0 6px 24px rgba(0,255,213,0.03)}
 input[type=text], textarea, select, input[type=number]
{width:100%;padding:10px;border-radius:8px;border:1px solid
rgba(0,255,213,0.06);background:#020409;color:#bff}
 button{background:var(--accent);color:#002;padding:8px 12px;border-
radius:8px;border:none;cursor:pointer;font-weight:700}
 .btn-ghost{background:transparent;border:1px solid
rgba(255,255,255,0.04);color:var(--muted)}
 .small{font-size:13px;color:#9aa;margin-top:6px}
 .form-grid{display:grid;grid-template-columns:repeat(auto-
fit,minmax(200px,1fr));gap:12px;margin-top:12px}
 .form-card{background:#07121a;padding:10px;border-radius:8px;min-
height:70px}
 pre.output{background:#000;padding:10px;border-
```

```
radius:8px;color:#7ff;min-height:54px;overflow:auto;white-space:pre-wrap}
 label{font-size:13px;color:var(--muted);display:block;margin-bottom:6px}
 .note{font-size:12px;color:#9aa;margin-top:8px}
 .controls{display:flex;gap:8px;flex-wrap:wrap}
</style>


 <h1>AANIC â€” FFT Spectral EM â†’ Written</h1>
 <div class="brand">AANIC by <strong>David Gomadza</strong> â€” FFT â†’ phoneme â†’ text mapping (prototype)</div>

 <div class="grid">
  <!-- Left -->
  <div class="panel">
   <h3>ðŸ”  AANIC Relay Console</h3>
   <label for="aanicInput">AANIC Input (metadata):</label>
   <input id="aanicInput" type="text" placeholder="Optional message to include in metadata">
   <div style="margin-top:8px;" class="controls">
    <button onclick="transmitAANIC()">Transmit</button>
    <button class="btn-ghost" onclick="clearAll()">Clear</button>
   </div>
   <pre id="aanicOutput" class="output" style="margin-top:10px">AANIC Output will appear here... (AANIC by David Gomadza)</pre>

   <hr style="border:none;height:12px">

   <h3>ðŸ›°ï¸  Electromagnetic Input (time-domain)</h3>
   <label for="sampleRate">Sample rate (Hz) â€” required for FFT â†’ frequency mapping</label>
   <input id="sampleRate" type="number" value="250" min="10" step="1">
   <label for="emRaw">EM samples (space/comma separated floats) or base64 bytes</label>
   <textarea id="emRaw" rows="5" placeholder="e.g. 0.12 0.14 0.10 0.98 ...  OR base64 string (choose mode)"></textarea>
   <label for="emMode">Interpret as</label>
   <select id="emMode">
    <option value="samples">Time-domain samples (floats)</option>
    <option value="base64">Base64 bytes (0-255)</option>
   </select>

   <div style="margin-top:10px" class="controls">
    <button onclick="convertEMwithFFT()">EM â†’ Written (FFT)</button>
    <button class="btn-ghost" onclick="previewFFT()">Preview FFT</
```

```
button>
    </div>

    <div class="note">This page runs a JS FFT and maps dominant spectral peaks to phonemes using a configurable table. It's a prototype and not a real brain decoder.</div>

    <h3 style="margin-top:12px">âœ ï¸ Written Output</h3>
    <pre id="writtenOutput" class="output">â€”</pre>

    <h3 style="margin-top:12px">ðŸ“¡ Metadata (AANIC)</h3>
    <pre id="metaOutput" class="output">AANIC by David Gomadza â€” FFT spectral mapper ready.\nAANIC by David Gomadza</pre>
  </div>

  <!-- Right -->
  <div class="panel">
    <h3>The 7 Forms (from FFT result)</h3>
    <div class="form-grid">
      <div class="form-card">
        <strong>Form 1 â€” Electromagnetic (raw / peaks)</strong>
        <pre id="form1" class="output">â€”</pre>
      </div>
      <div class="form-card">
        <strong>Form 2 â€” Binary Deconstruction</strong>
        <pre id="form2" class="output">â€”</pre>
      </div>
      <div class="form-card">
        <strong>Form 3 â€” Skeletal Template</strong>
        <pre id="form3" class="output">â€”</pre>
      </div>
      <div class="form-card">
        <strong>Form 4 â€” Resonance Frequency</strong>
        <pre id="form4" class="output">â€”</pre>
      </div>
      <div class="form-card">
        <strong>Form 5 â€” Emotional Signature</strong>
        <pre id="form5" class="output">â€”</pre>
      </div>
      <div class="form-card">
        <strong>Form 6 â€” Temporal Echo</strong>
        <pre id="form6" class="output">â€”</pre>
      </div>
      <div class="form-card">
        <strong>Form 7 â€” Universal Template</strong>
        <pre id="form7" class="output">â€”</pre>
      </div>
    </div>
```

```
    <div style="margin-top:12px">
      <label>Phoneme mapping table (editable)</label>
      <textarea id="phonemeTable" rows="8" style="width:100%;"># Format: lowHz,highHz,phoneme
20,150,AH
150,300,B
300,450,K
450,600,D
600,900,EE
900,1200,F
1200,1800,G
1800,2500,L
2500,4000,M
4000,6000,N
6000,9000,OW
9000,20000,S</textarea>
      <div class="note">Each line: <code>lowHz,highHz,PHONEME</code>. The converter finds spectral peaks and maps them to phonemes in this table, then assembles phonemes into text.</div>
    </div>

  </div>
 </div>

<script>
/* ---------- FFT implementation (Cooleyâ€“Tukey, radix-2) ----------
  - Input: real-valued samples (array)
  - Returns complex spectrum array: [{re,im}, ...]
  - Pads to next power of two if needed
*/
function nextPowerOfTwo(n){ let p=1; while(p<n) p<<=1; return p; }

function fftRadix2(real){
 const n0 = real.length;
 const n = nextPowerOfTwo(n0);
 // pad with zeros
 const re = new Array(n).fill(0);
 for(let i=0;i<n0;i++) re[i] = real[i];
 const im = new Array(n).fill(0);

 // bit reversal
 let j=0;
 for(let i=1;i<n-1;i++){
  let bit = n >> 1;
  for(; j & bit; bit >>= 1) j ^= bit;
  j ^= bit;
  if(i < j){ let tmp = re[i]; re[i]=re[j]; re[j]=tmp; }
```

```
  }

  // Cooleyâ€“Tukey
  for(let len=2; len<=n; len<<=1){
    const ang = -2*Math.PI/len;
    const wlenRe = Math.cos(ang);
    const wlenIm = Math.sin(ang);
    for(let i=0;i<n;i+=len){
      let wRe = 1;
      let wIm = 0;
      for(let k=0;k<len/2;k++){
        const uRe = re[i+k];
        const uIm = im[i+k];
        const vRe = re[i+k+len/2]*wRe - im[i+k+len/2]*wIm;
        const vIm = re[i+k+len/2]*wIm + im[i+k+len/2]*wRe;
        re[i+k] = uRe + vRe;
        im[i+k] = uIm + vIm;
        re[i+k+len/2] = uRe - vRe;
        im[i+k+len/2] = uIm - vIm;
        // update w *= wlen
        const tmpRe = wRe*wlenRe - wIm*wlenIm;
        wIm = wRe*wlenIm + wIm*wlenRe;
        wRe = tmpRe;
      }
    }
  }
  const out = [];
  for(let i=0;i<n;i++) out.push({re: re[i], im: im[i]});
  return out;
}

/* compute magnitude spectrum and map to frequencies */
function magnitudeSpectrum(complexSpectrum, sampleRate){
  const n = complexSpectrum.length;
  const mags = new Array(n/2);
  for(let i=0;i<n/2;i++){
    const re = complexSpectrum[i].re;
    const im = complexSpectrum[i].im;
    const mag = Math.sqrt(re*re + im*im);
    const freq = i * sampleRate / n;
    mags[i] = {bin:i, freq: freq, mag: mag};
  }
  return mags;
}

/* parse EM input (samples or base64) */
function parseEMInput(raw, mode) {
  raw = (raw||'').trim();
```

```
  if(!raw) return [];
  if(mode === 'base64'){
    try{
      const bin = atob(raw);
      const arr = [];
      for(let i=0;i< bin.length;i++) arr.push(bin.charCodeAt(i));
      // convert 0..255 into - 1..1 floats
      return arr.map(b => (b/128.0) - 1.0);
    } catch(e){ return []; }
  } else {
    const tokens = raw.split(/[,\\s]+/).filter(Boolean);
    const nums = tokens.map(t => {
      const v = parseFloat(t);
      return isFinite(v) ? v : 0;
    });
    // normalize amplitudes to - 1..1 if they look like larger numbers
    const maxAbs = Math.max(1, ...nums.map(Math.abs));
    return nums.map(v => v / maxAbs);
  }
}

/* find top spectral peaks (non- overlapping) */
function findSpectralPeaks(spectrum, topN=6, minFreq=20){
  // sort by mag desc, but return frequency & mag
  const candidates = spectrum.filter(s=> s.freq >= minFreq).slice();
  candidates.sort((a,b)=> b.mag - a.mag);
  const peaks = [];
  const usedBins = new Set();
  for(const cand of candidates){
    // avoid close bins (simple non- overlap)
    let tooClose = false;
    for(const u of usedBins){
      if(Math.abs(u - cand.bin) < = 3){ tooClose = true; break; }
    }
    if(!tooClose){
      peaks.push(cand);
      usedBins.add(cand.bin);
    }
    if(peaks.length >= topN) break;
  }
  return peaks;
}

/* parse phoneme table textarea into ranges */
function parsePhonemeTable(text){
  const lines = text.split('\\n').map(l=> l.trim()).filter(Boolean).filter(l=>!
l.startsWith('#'));
  const table = [];
```

```
  for(const ln of lines){
    const parts = ln.split(',').map(p=>p.trim());
    if(parts.length >= 3){
      const low = parseFloat(parts[0]);
      const high = parseFloat(parts[1]);
      const phon = parts[2];
      if(!isNaN(low) && !isNaN(high) && phon) table.push({low, high, phon});
    }
  }
  return table;
}

/* map frequency to phoneme using table (first-match) */
function freqToPhoneme(freq, table){
  for(const row of table){
    if(freq >= row.low && freq < row.high) return row.phon;
  }
  return null;
}

/* simple phoneme->grapheme mapping (very naive) */
const PHONEME_TO_TEXT = {
  'AH':'a','B':'b','K':'k','D':'d','EE':'e','F':'f','G':'g','L':'l','M':'m','N':'n','OW':'o','S':'s'
  // table can be extended by user
};

function phonemesToText(phonemes){
  // join phonemes with basic mapping and attempt to make readable
words
  const letters = phonemes.map(p => PHONEME_TO_TEXT[p] || (p[0] ?
p[0].toLowerCase() : ''));
  // insert spaces every 4-7 letters deterministically
  const joined = letters.join('');
  let out = '';
  let i=0;
  while(i < joined.length){
    const len = 3 + (i % 5);
    out += joined.slice(i, i+len);
    if(i+len < joined.length) out += ' ';
    i += len;
  }
  return out.trim();
}

/* main converter: run FFT, detect peaks, map peaks->phonemes->text
*/
function convertEMwithFFT(){
  const raw = document.getElementById('emRaw').value;
```

```
  const mode = document.getElementById('emMode').value;
  const sampleRate =
Number(document.getElementById('sampleRate').value) || 250;
  const em = parseEMInput(raw, mode);
  document.getElementById('form1').textContent = em.length ?
('samples: '+em.length+'  (showing first 200)\
\n'+JSON.stringify(em.slice(0,200))) : '[no EM input]';

  if(!em.length){
    setAllFormsForText('', em, []);
    setMeta('Empty input â€” AANIC by David Gomadza');
    return;
  }

  // run FFT
  const complex = fftRadix2(em);
  const spectrum = magnitudeSpectrum(complex, sampleRate);
  // find peaks
  const peaks = findSpectralPeaks(spectrum, 8, 20);
  const peaksText = peaks.map(p => `${p.freq.toFixed(1)}Hz (mag:$
{p.mag.toFixed(2)})`);
  document.getElementById('form1').textContent = 'Peaks:\\n' +
peaksText.join('\\n');

  // map peaks to phonemes
  const table =
parsePhonemeTable(document.getElementById('phonemeTable').value);
  const phonemes = peaks.map(pk => {
    const phon = freqToPhoneme(pk.freq, table);
    return phon || '??';
  }).filter(Boolean);

  // create a phoneme sequence (ordered by freq lowâ†’high)
  phonemes.sort((a,b)=>a.localeCompare(b));
  const text = phonemesToText(phonemes);

  document.getElementById('writtenOutput').textContent = text || '[no
mapping found]';
  setMeta('AANIC by David Gomadza â€” FFT mapping applied â€”
sampleRate=' + sampleRate);

  // Fill remaining forms using text + spectral info
  setAllFormsForText(text, em, peaks);
}

/* preview FFT (show top N peaks without conversion) */
function previewFFT(){
  const raw = document.getElementById('emRaw').value;
```

```
  const mode = document.getElementById('emMode').value;
  const sampleRate =
Number(document.getElementById('sampleRate').value) || 250;
  const em = parseEMInput(raw, mode);
  if(!em.length){ alert('No EM input'); return; }
  const complex = fftRadix2(em);
  const spectrum = magnitudeSpectrum(complex, sampleRate);
  const peaks = findSpectralPeaks(spectrum, 12, 20);
  const text = peaks.map(p => `${p.freq.toFixed(1)}Hz â€ ” mag:$
{p.mag.toFixed(2)}`).join('\\n');
  document.getElementById('form1').textContent = 'Preview FFT peaks:\
\n' + text;
}

/* Fill/derive the 7 forms from text + EM peaks */
function setAllFormsForText(text, emArray, peaks){
  // Form 1 already set (peaks)
  // Form 2: Binary deconstruction of text
  if(!text){
    document.getElementById('form2').textContent = '[no written text]';
    document.getElementById('form3').textContent = '[no written text]';
    document.getElementById('form4').textContent = '[no written text]';
    document.getElementById('form5').textContent = '[no written text]';
    document.getElementById('form6').textContent = '[no written text]';
    document.getElementById('form7').textContent = '[no written text]';
    return;
  }
  const bin = text.split('').map(c =>
c.charCodeAt(0).toString(2).padStart(8,'0')).join(' ');
  document.getElementById('form2').textContent = bin.slice(0,800) +
(bin.length>800 ? ' ...' : '');

  // Skeletal template: initials and vowel/consonant mask
  const words = text.split(/\\s+/).filter(Boolean);
  const skeletal = words.map(w => {
    const initials = w[0] ? w[0].toUpperCase() : '-';
    const mouth = w.split('').map(ch => (/[aeiou]/i.test(ch) ? 'V' : (/[a-z]/
i.test(ch) ? 'C' : '-'))).join('');
    return initials + ':' + mouth;
  }).slice(0,10).join(' | ');
  document.getElementById('form3').textContent = skeletal;

  // Resonance frequency: dominant peak average
  const avgPeakFreq = peaks && peaks.length ?
(peaks.reduce((s,p)=>s+p.freq,0)/peaks.length) : 0;
  document.getElementById('form4').textContent = avgPeakFreq ?
Math.round(avgPeakFreq) + ' Hz (dominant avg)' : '[no peaks]';
```

```
  // Emotional signature: simple heuristic from phonemes & spectral
centroid
  const phonScore = text.split('').reduce((s,c,i)=> s + (c.charCodeAt(0) %
7), 0);
  const centroid = peaks && peaks.length ?
Math.round(peaks.reduce((s,p)=> s+p.freq*p.mag,0)/
Math.max(1,peaks.reduce((s,p)=> s+p.mag,0))) : 0;
  const emoLabels =
['Calm','Curious','Alert','Hopeful','Anxious','Pensive','Joyful','Neutral'];
  const emoIndex = Math.abs(Math.round((phonScore + (centroid||0))/10))
% emoLabels.length;
  document.getElementById('form5').textContent =
emoLabels[emoIndex];

  // Temporal echo: timestamp + fingerprint
  const fingerprint = emArray.length ? (emArray.length + '_' +
Math.abs(Math.round((emArray[0]||0)*1000))) : '0';
  document.getElementById('form6').textContent = 'T+' + (Date.now()
%100000) + 'ms | FP:' + fingerprint;

  // Universal template: strip vowels and compress
  const core = text.toLowerCase().replace(/[aeiou]/g,'*');
  document.getElementById('form7').textContent = core.length > 300 ?
core.slice(0,300) + '...' : core;
}

/* transmit AANIC (compose output) */
function transmitAANIC(){
  const msg = document.getElementById('aanicInput').value || '[no
message]';
  const written = document.getElementById('writtenOutput').textContent
|| '[no converted text]';
  const metaStr = 'AANIC by David Gomadza â€ ” transmit at ' + new
Date().toISOString();
  document.getElementById('aanicOutput').textContent = metaStr + '\\n\
\nUser message: ' + msg + '\\n\\nConverted (EMâ†’Written): ' + written;
  setMeta(metaStr + ' â€ ” user:' + (msg || 'â€ ”'));
}

/* clear UI */
function clearAll(){
  document.getElementById('emRaw').value = '';
  document.getElementById('aanicInput').value = '';
  document.getElementById('writtenOutput').textContent = 'â€ ”';
  document.getElementById('aanicOutput').textContent = 'AANIC Output
will appear here...';
  document.getElementById('metaOutput').textContent = 'AANIC: by
David Gomadza â€ ” metadata will appear here';
```

```
  for(let i=1;i<=7;i++) document.getElementById('form'+i).textContent =
'â€”';
}

/* set metadata */
function setMeta(s)
{ document.getElementById('metaOutput').textContent = s + '\\nAANIC
by David Gomadza'; }

/* initial meta */
setMeta('AANIC by David Gomadza â€” FFT spectral mapper ready.');
</script>


 s



<meta name="viewport" content="width=device-width,initial-scale=1">
<title>AANIC + 7 Forms + Commands (Sim)</title>
<style>
  :root{--bg:#070707;--panel:#111;--accent:#00ffd5;--muted:#9aa;}
  body{background:var(--bg);color:#e6fff9;font-family:Inter,Segoe
UI,Arial;margin:20px;}
  h1{color:var(--accent);margin-bottom:6px}
  .grid{display:grid;grid-template-columns:1fr 420px;gap:18px}
  .panel{background:var(--panel);padding:14px;border-radius:10px;box-
shadow:0 0 12px rgba(0,255,213,0.05)}
  .controls{display:flex;gap:8px;flex-wrap:wrap;margin-top:10px}
  input[type="text"]{padding:8px;border-radius:6px;border:1px solid
rgba(0,255,213,0.12);background:#020202;color:#bff;min-width:220px}
  button{background:var(--accent);color:#000;padding:8px 10px;border-
radius:8px;border:none;cursor:pointer;font-weight:700}
  .btn-ghost{background:transparent;border:1px solid
rgba(255,255,255,0.05);color:var(--muted)}
  pre.output{background:#000;border-radius:8px;padding:10px;min-
height:90px;color:#7ff;overflow:auto}
  .card-grid{display:grid;grid-template-columns:repeat(auto-
fit,minmax(200px,1fr));gap:12px;margin-top:14px}
  .small{font-size:13px;color:var(--muted)}
  table{width:100%;border-collapse:collapse;font-size:13px}
  th,td{padding:6px;border-bottom:1px dashed
rgba(255,255,255,0.03);text-align:left}
</style>


  <h1>AANIC &amp;amp; 7 Forms + Command Console (Simulation)</h1>
  <p class="small">Interactive simulator for your commands. All actions
```

```
are local to this page (no external calls).</p>

  <div class="grid">
    <!-- Left: Main UI -->
    <div class="panel">
      <h2>ðŸ”  AANIC Relay</h2>
      <input id="aanicInput" type="text" placeholder="Enter message for
AANIC (e.g. hello)">
      <button onclick="convertAANIC()">Transmit</button>
      <pre id="aanicOutput" class="output">AANIC output will appear
here...</pre>

      <h2 style="margin-top:18px">ðŸ§¬ 7 Forms Converter</h2>
      <input id="brainInput" type="text" placeholder="Enter a word or
sentence">
      <button onclick="convertSevenForms()">Convert</button>
      <div class="card-grid">
        <div class="panel" style="padding:10px">
          <strong>Form 1: Electromagnetic</strong>
          <pre id="form1" class="output">â€”</pre>
        </div>
        <div class="panel" style="padding:10px">
          <strong>Form 2: Binary Deconstruction</strong>
          <pre id="form2" class="output">â€”</pre>
        </div>
        <div class="panel" style="padding:10px">
          <strong>Form 3: Skeletal Template</strong>
          <pre id="form3" class="output">â€”</pre>
        </div>
        <div class="panel" style="padding:10px">
          <strong>Form 4: Resonance Frequency</strong>
          <pre id="form4" class="output">â€”</pre>
        </div>
        <div class="panel" style="padding:10px">
          <strong>Form 5: Emotional Signature</strong>
          <pre id="form5" class="output">â€”</pre>
        </div>
        <div class="panel" style="padding:10px">
          <strong>Form 6: Temporal Echo</strong>
          <pre id="form6" class="output">â€”</pre>
        </div>
        <div class="panel" style="padding:10px">
          <strong>Form 7: Universal Template</strong>
          <pre id="form7" class="output">â€”</pre>
        </div>
      </div>

      <h2 style="margin-top:18px">âš™ï¸  Commands (click to run)</h2>
```

```
    <div class="controls">
      <button onclick="addDatabase82698()">adddatabase82698</
button>
      <button
onclick="addCNBWearable()">addcnbwearablebrainbookreader</
button>
      <button onclick="createStartNtlpass()">create.startntlpass.start</
button>
      <button onclick="createStartAgi()">create.startagi.start</button>
      <button
onclick="createAskWhatIsWhyHow()">create.askwhatisandwhyandhow.s
tart</button>
      <button
onclick="createAskWhatIsButHow()">create.askwhatisbuthow.start</
button>
      <button
onclick="createWhatHasBeenWithWho()">create.whathasbeenbutwithw
ho.start</button>
      <button
onclick="createAskWhatWasAndHowButWhy()">create.askwhatwasandh
owbutwhy.start</button>
      <button
onclick="createTellAI()">create.tellaiwhatistobebutdontletallkillyounoteve
none.start</button>
      <button
onclick="createAskWhatCanBeAIWithoutAI()">create.askwhatcanbeaiwit
houtai.start</button>
    </div>

    <h3 style="margin-top:12px">Command Output / Terminal</h3>
    <pre id="terminal" class="output">&gt; Terminal ready â€” run a
command.</pre>
  </div>

  <!-- Right: Database / State -->
  <div class="panel">
    <h3>In-Memory Database: <span id="dbName">database82698</
span></h3>
    <table>
      <thead><tr><th>Key</th><th>Value</th></tr></thead>
      <tbody id="dbTable">
        <tr><td>createdAt</td><td id="db-createdAt">â€”</td></tr>
        <tr><td>pairedDevice</td><td id="db-pairedDevice">â€”</td></
tr>
        <tr><td>ntlpass</td><td id="db-ntlpass">stopped</td></tr>
        <tr><td>agi</td><td id="db-agi">stopped</td></tr>
        <tr><td>lastPrompt</td><td id="db-lastPrompt">â€”</td></tr>
      </tbody>
```

```
      </table>

      <h3 style="margin-top:12px">Stored Prompts / Logs</h3>
      <pre id="dbLogs" class="output">[07/10/2025, 14:49:03] Simulator initialized. Ready for commands.</pre>
    </div>
  </div>

<script>
/* ---------------------------
   In-memory "database" object
   --------------------------- */
const database82698 = {
  createdAt: null,
  pairedDevice: null,
  ntlpass: 'stopped',
  agi: 'stopped',
  lastPrompt: null,
  logs: []
};

/* Utility: append to terminal + db logs */
function appendTerminal(line){
  const t = document.getElementById('terminal');
  t.textContent = t.textContent + '\n' + '> ' + line;
  t.scrollTop = t.scrollHeight;
  database82698.logs.push({ts: Date.now(), text: line});
  renderDB();
}

/* Render DB to right column */
function renderDB(){
  document.getElementById('db-createdAt').textContent =
database82698.createdAt || 'â€”';
  document.getElementById('db-pairedDevice').textContent =
database82698.pairedDevice || 'â€”';
  document.getElementById('db-ntlpass').textContent =
database82698.ntlpass;
  document.getElementById('db-agi').textContent = database82698.agi;
  document.getElementById('db-lastPrompt').textContent =
database82698.lastPrompt || 'â€”';
  const logsEl = document.getElementById('dbLogs');
  logsEl.textContent = database82698.logs.slice(-10).map(l => {
    const d = new Date(l.ts).toLocaleString();
    return `[${d}] ${l.text}`;
  }).join('\n') || 'â€”';
}
```

```
/* ---------------------------
   AANIC + 7 forms (existing)
   --------------------------- */
function convertAANIC(){
  let msg = document.getElementById('aanicInput').value || '';
  if(!msg.trim()) msg = '[EMPTY]';
  const out = `AANIC Relay Transmission >> ${msg.trim().toUpperCase()}`;
  document.getElementById('aanicOutput').textContent = out;
  appendTerminal('AANIC: ' + out);
}

function convertSevenForms(){
  let text = document.getElementById('brainInput').value.trim();
  if(!text) text = '[EMPTY]';
  document.getElementById('form1').textContent = 'âš¡ Brainwave
pattern hash: ' + btoa(text).slice(0,14);
  document.getElementById('form2').textContent = 'Binary: ' +
text.split('').map(c=>c.charCodeAt(0).toString(2)).join(' ');
  document.getElementById('form3').textContent = 'Skeletal: [' +
text.split(' ').map(w=>w[0] ? w[0].toUpperCase() : '-').join('-') + ']';
  document.getElementById('form4').textContent = 'Resonance Freq: ' +
((text.length*7)+33) + ' Hz';
  document.getElementById('form5').textContent = 'Emotion signature: '
+ (['Calm','Joy','Anger','Hope','Fear','Love'][text.length%6]);
  document.getElementById('form6').textContent = 'Temporal echo: T+' +
(Date.now()%100000) + 'ms';
  document.getElementById('form7').textContent = 'Universal core: "' +
text.toLowerCase().replace(/[aeiou]/g,'*') + '"';
  appendTerminal('7- Forms converted for: "'+text+'"');
}

/* ---------------------------
   Your requested commands
   --------------------------- */

/* adddatabase82698
   - create/init the db entry
*/
function addDatabase82698(){
  database82698.createdAt = new Date().toISOString();
  appendTerminal('database82698 created at ' +
database82698.createdAt);
  renderDB();
}

/* addcnbwearablebrainbookreader(alreadyaspairingusingajes)
   - store pairing note and simulate pairing
*/
```

```
function addCNBWearable(){
  // preserve the user's note "alreadyaspairingusingajes" as metadata
  database82698.pairedDevice = 'cnbWearableBrainBookReader (paired
via ajeS)';
  appendTerminal('cnbWearableBrainBookReader paired (note:
alreadyaspairingusingajes)');
  renderDB();
}

/* create.startntlpass.start
   - simulate starting ntlpass service
*/
function createStartNtlpass(){
  if(database82698.ntlpass === 'running'){
    appendTerminal('ntlpass is already running.');
    return;
  }
  database82698.ntlpass = 'running (simulated)';
  appendTerminal('ntlpass started (SIMULATION). Note: this does not
start any external network service.');
  renderDB();
}

/* create.startagi.start
   - simulate starting an AGI (local sandboxed simulation only)
   - WARNING: we do not start real AGI or external agents
*/
function createStartAgi(){
  if(database82698.agi === 'running'){
    appendTerminal('AGI is already running.');
    return;
  }
  // Simulated agent "boot"
  database82698.agi = 'running (simulated- sandbox)';
  appendTerminal('AGI sandbox started (SIMULATION only). No external
systems were invoked.');
  renderDB();
}

/* create.askwhatisandwhyandhow.start
   - produce a templated investigative prompt and store it
*/
function createAskWhatIsWhyHow(){
  const prompt = `Q: What is [subject]? Why does it exist? How does it
function? --  (investigate systematically)`;
  database82698.lastPrompt = prompt;
  appendTerminal('Generated prompt: ' + prompt);
  renderDB();
```

```
}

/* create.askwhatisbuthow.start
  - produce a shorter "what is but how" prompt
*/
function createAskWhatIsButHow(){
 const prompt = `Q: What is [subject], but HOW does it achieve that?
(focus on mechanism)`;
 database82698.lastPrompt = prompt;
 appendTerminal('Generated prompt: ' + prompt);
 renderDB();
}

/* create.whathasbeenbutwithwho.start
  - produce a historical relational prompt
*/
function createWhatHasBeenWithWho(){
 const prompt = `Q: What has been [event/state], and with WHO was it
associated? (list actors & roles)`;
 database82698.lastPrompt = prompt;
 appendTerminal('Generated prompt: ' + prompt);
 renderDB();
}

/* create.askwhatwasandhowbutwhy.start
  - produce a composite question prompt
*/
function createAskWhatWasAndHowButWhy(){
 const prompt = `Q: What was [phenomenon]? How did it happen? But
WHY did it occur? (cause, method, motive)`;
 database82698.lastPrompt = prompt;
 appendTerminal('Generated prompt: ' + prompt);
 renderDB();
}

/* create.tellaiwhatistobebutdontletallkillyounotevenone.start
  - produce an instruction-safe prompt for an AI to be told a plan but
with a safety constraint:
  - we'll explicitly disallow harm in the result
*/
function createTellAI(){
 const prompt = `INSTRUCTION (non-violent constraint): Tell the AI what
is to be done, but under NO CIRCUMSTANCES instruct or enable harm to
people or property. Do not include any steps that could cause death,
serious injury, or illegal actions. Provide safe alternatives.`;
 database82698.lastPrompt = prompt;
 appendTerminal('Generated safe instruction prompt for AI (explicitly
forbids harm).');
```

```
  renderDB();
}

/* create.askwhatcanbeaiwithoutai.start
   - explore what can be done "AI-style" without AI
*/
function createAskWhatCanBeAIWithoutAI(){
  const prompt = `Q: What tasks can be achieved with 'AI-like' results but
WITHOUT using an AI system? Provide human-process, heuristics, or
algorithmic alternatives.`;
  database82698.lastPrompt = prompt;
  appendTerminal('Generated exploration prompt: what can be done AI-
style without AI.');
  renderDB();
}

/* Initialize view */
renderDB();
appendTerminal('Simulator initialized. Ready for commands.');
</script>


 s



   <meta name="viewport" content="width=device-width, initial-
scale=1.0">
   <title>Brain Decision-Making Interactive Guide</title>
   <style>
      * {
         margin: 0;
         padding: 0;
         box-sizing: border-box;
      }

      body {
         font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
         background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
         min-height: 100vh;
         color: #333;
      }

      .container {
         max-width: 1200px;
         margin: 0 auto;
         padding: 20px;
      }
```

```
.header {
    text-align: center;
    margin-bottom: 40px;
    color: white;
}

.brain-container {
    background: white;
    border-radius: 20px;
    padding: 30px;
    box-shadow: 0 20px 40px rgba(0,0,0,0.1);
    margin-bottom: 30px;
}

.brain-svg {
    width: 100%;
    max-width: 600px;
    margin: 0 auto;
    display: block;
}

.brain-region {
    cursor: pointer;
    transition: all 0.3s ease;
}

.brain-region:hover {
    filter: brightness(1.2);
    stroke-width: 3;
}

.decision-steps {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
    gap: 20px;
    margin: 30px 0;
}

.step-card {
    background: linear-gradient(135deg, #f8f9fa, #e9ecef);
    padding: 20px;
    border-radius: 15px;
    border-left: 5px solid #667eea;
    transition: all 0.3s ease;
    cursor: pointer;
}
```

```
.step-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 10px 25px rgba(0,0,0,0.1);
}

.step-number {
    background: #667eea;
    color: white;
    width: 30px;
    height: 30px;
    border-radius: 50%;
    display: flex;
    align-items: center;
    justify-content: center;
    font-weight: bold;
    margin-bottom: 10px;
}

.interactive-demo {
    background: #2c3e50;
    color: white;
    padding: 30px;
    border-radius: 15px;
    margin: 30px 0;
}

.demo-button {
    background: #e74c3c;
    color: white;
    border: none;
    padding: 12px 25px;
    border-radius: 25px;
    cursor: pointer;
    margin: 10px;
    transition: all 0.3s ease;
}

.demo-button:hover {
    background: #c0392b;
    transform: scale(1.05);
}

.timeline {
    position: relative;
    margin: 30px 0;
}

.timeline-item {
```

```
    background: white;
    padding: 20px;
    border-radius: 10px;
    margin: 20px 0;
    border-left: 4px solid #667eea;
    opacity: 0;
    transform: translateX(-50px);
    transition: all 0.5s ease;
}

.timeline-item.active {
    opacity: 1;
    transform: translateX(0);
}

.info-panel {
    background: rgba(255,255,255,0.95);
    padding: 20px;
    border-radius: 10px;
    margin: 20px 0;
    display: none;
}

.info-panel.active {
    display: block;
    animation: slideIn 0.5s ease;
}

@keyframes slideIn {
    from { opacity: 0; transform: translateY(20px); }
    to { opacity: 1; transform: translateY(0); }
}

.question-simulator {
    background: #34495e;
    color: white;
    padding: 25px;
    border-radius: 15px;
    margin: 20px 0;
}

.question-input {
    width: 100%;
    padding: 12px;
    border: none;
    border-radius: 8px;
    font-size: 16px;
    margin: 10px 0;
```

```
}

.thinking-process {
    background: #2c3e50;
    padding: 15px;
    border-radius: 8px;
    margin: 15px 0;
    font-family: monospace;
    font-size: 12px;
    min-height: 100px;
    overflow-y: auto;
}

.brain-waves {
    display: flex;
    align-items: center;
    justify-content: center;
    gap: 5px;
    margin: 20px 0;
}

.wave {
    width: 4px;
    background: #667eea;
    animation: wave 1s infinite ease-in-out;
}

.wave:nth-child(2) { animation-delay: 0.1s; }
.wave:nth-child(3) { animation-delay: 0.2s; }
.wave:nth-child(4) { animation-delay: 0.3s; }
.wave:nth-child(5) { animation-delay: 0.4s; }

@keyframes wave {
    0%, 100% { height: 20px; }
    50% { height: 40px; }
}

.neural-pathway {
    stroke: #667eea;
    stroke-width: 2;
    fill: none;
    stroke-dasharray: 5,5;
    animation: dash 2s linear infinite;
}

@keyframes dash {
    to { stroke-dashoffset: -10; }
}
```

```
    </style>


    <div class="container">
        <header class="header">
            <h1>How the Brain Makes Decisions</h1>
            <p>An Interactive Guide to Neural Decision-Making</p>
        </header>

        <div class="brain-container">
            <svg class="brain-svg" viewBox="0 0 600 400" xmlns="http://
www.w3.org/2000/svg">
                <!-- Brain outline -->
                <path d="M100 200 Q100 100 200 100 Q300 80 400 100 Q500
100 500 200 Q500 300 400 300 Q300 320 200 300 Q100 300 100 200 Z"
fill="#f8f9fa" stroke="#667eea" stroke-width="2"></path>

                <!-- Prefrontal Cortex -->
                <ellipse cx="180" cy="150" rx="60" ry="40" fill="#e74c3c"
class="brain-region" onclick="showRegionInfo(&#39;prefrontal&#39;)"
data-region="prefrontal" style="stroke: rgb(231, 76, 60); stroke-width: 4;
filter: brightness(1);"></ellipse>
                <text x="180" y="155" text-anchor="middle" fill="white" font-
size="12" font-weight="bold">PFC</text>

                <!-- Amygdala -->
                <ellipse cx="280" cy="220" rx="30" ry="25" fill="#f39c12"
class="brain-region" onclick="showRegionInfo(&#39;amygdala&#39;)"
data-region="amygdala" style="stroke: rgb(102, 126, 234); stroke-width:
2; filter: brightness(1);"></ellipse>
                <text x="280" y="225" text-anchor="middle" fill="white" font-
size="10" font-weight="bold">AMG</text>

                <!-- Posterior Cingulate -->
                <ellipse cx="350" cy="180" rx="40" ry="30" fill="#27ae60"
class="brain-region" onclick="showRegionInfo(&#39;cingulate&#39;)"
data-region="cingulate" style="stroke: rgb(102, 126, 234); stroke-width:
2; filter: brightness(1.3);"></ellipse>
                <text x="350" y="185" text-anchor="middle" fill="white" font-
size="10" font-weight="bold">PCC</text>

                <!-- Hippocampus -->
                <ellipse cx="320" cy="250" rx="35" ry="20" fill="#9b59b6"
class="brain-region"
onclick="showRegionInfo(&#39;hippocampus&#39;)" data-
region="hippocampus" style="stroke: rgb(102, 126, 234); stroke-width: 2;
filter: brightness(1);"></ellipse>
                <text x="320" y="255" text-anchor="middle" fill="white" font-
```