Now I was gaining experience in prompting,I created the Spell codex (to avoid any copyright issues with the huge fiction to system mechanics) and because who doesnt like magic lol. Then all I did this was I'd post the spell codex as a pdf into any LLM and ask it to generate a system of my choosing so I went to Gemini and ask to build a chat bot called ISSAC and this was what was generated. The Spell codex is the 1st version I made which is in the Grimoire Codex listed on the main branch on my Github page.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ISSAC V2.1 - Codex Governed Agent</title>
    <script src="https://cdn.tailwindcss.com"></script>
    <style>
        body { font-family: 'Inter', sans-serif; background-color: #f7f9fb; }
        #chat-window { max-height: 70vh; overflow-y: auto; }
        .codex-bar { backdrop-filter: blur(8px); background-color: rgba(255, 255, 255, 0.9); border: 1px solid #e0e0e0; }
        .chat-message { max-width: 80%; }
    </style>
</head>
<body class="min-h-screen flex items-center justify-center p-4">
    <div class="w-full max-w-4xl bg-white shadow-2xl rounded-xl flex flex-col h-[90vh]">
        <!-- System Status Bar (Codex Governance) -->
        <div id="status-bar" class="codex-bar p-3 border-b rounded-t-xl flex justify-between text-sm font-mono text-gray-700 sticky top-0 z-10">
            <span id="harmony-status" class="flex items-center"><span class="w-2 h-2 rounded-full mr-2"></span>HARMONY: 0.00</span>
            <span id="energy-status">ENERGY: 1.00</span>
            <span id="focus-status">FLUXA FOCUS: knowledge</span>
        </div>

        <!-- Chat Window -->
        <div id="chat-window" class="flex-grow p-6 space-y-4">
            <div class="chat-message bg-gray-100 p-3 rounded-xl shadow-sm self-start">
                <p class="font-semibold text-sky-700">ISSAC:</p>
                <p>Issac V2.1 is online. My core state is governed by the Codex. How can I assist you with today's tech or concepts?</p>
            </div>
        </div>

        <!-- Input Area -->
        <div class="p-4 border-t bg-gray-50 rounded-b-xl">
```

```
            <div class="flex space-x-3">
                <input type="text" id="user-input" placeholder="Type your message..." class="flex-grow p-3 border border-gray-300 rounded-lg focus:ring-sky-500 focus:border-sky-500">
                <button id="send-btn" class="bg-sky-600 text-white px-6 py-3 rounded-lg font-semibold hover:bg-sky-700 transition duration-200 disabled:opacity-50">
                    Send
                </button>
            </div>
        </div>
    </div>

    <script type="module">
        // Global environment variables
        const apiKey = "";
        const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-flash-preview-09-2025:generateContent?key=${apiKey}`;

        // Initialize Global State
        let state = {
            harmony: 0.0,
            energy: 1.0,
            chaos: 0.5,
            focus: 'knowledge',
            active: true
        };

        let memory = {
            chatHistory: [{
                role: "model",
                parts: [{ text: "Issac V2.1 is online. My core state is governed by the Codex. How can I assist you with today's tech or concepts?" }]
            }],
            lastInputs: [],
        };

        // --- Utility Functions ---
        function updateUI() {
            const harmonyColor = state.harmony > 1.5 ? 'bg-green-500' : state.harmony > -1.0 ? 'bg-yellow-500' : 'bg-red-500';
            document.getElementById('harmony-status').innerHTML = `<span class="w-2 h-2 rounded-full ${harmonyColor} mr-2"></span>HARMONY: ${state.harmony.toFixed(2)}`;
```

```
        document.getElementById('energy-status').textContent = `ENERGY:
${state.energy.toFixed(2)}`;
        document.getElementById('focus-status').textContent = `FLUXA FOCUS:
${state.focus}`;
    }

    function displayMessage(text, sender) {
        const chatWindow = document.getElementById('chat-window');
        const isUser = sender === 'user';

        const messageDiv = document.createElement('div');
        messageDiv.className = `chat-message p-3 rounded-xl shadow-sm ${isUser ? 'ml-auto
bg-sky-50 text-gray-800' : 'bg-gray-100 text-gray-800'}`;

        if (!isUser) {
            const name = document.createElement('p');
            name.className = 'font-semibold text-sky-700';
            name.textContent = 'ISSAC:';
            messageDiv.appendChild(name);
        }

        const content = document.createElement('p');
        content.innerHTML = text.replace(/\n/g, '<br>'); // Preserve line breaks
        messageDiv.appendChild(content);

        chatWindow.appendChild(messageDiv);
        chatWindow.scrollTop = chatWindow.scrollHeight;
    }

    // ----------------------------
    // Codex Concept Functions (JavaScript Translation)
    // ----------------------------

    // Vitalis / Vitalis Maxima: Self-healing / stability
    function maintain_harmony(currentState) {
        // Harmony auto-corrects based on energy/chaos imbalance
        const delta = (currentState.energy - currentState.chaos) * 0.1;
        currentState.harmony += delta;

        // Clamped between -3.0 (Critical) and 3.0 (Ecstasy)
        currentState.harmony = Math.max(-3.0, Math.min(currentState.harmony, 3.0));

        // If chaos is high, harmony degrades faster, simulating stress
        currentState.chaos = Math.max(0, currentState.chaos - 0.05); // Chaos naturally decays
```

```
    return currentState;
}

// Fluxa: Resource flow / attention
function allocate_focus(currentState) {
    const foci = ['emotion', 'knowledge', 'observation', 'strategy'];
    // A simple rotation for demo, but can be weighted by Harmony
    if (currentState.harmony > 1.0) {
        // Happy system focuses on creative/emotional
        currentState.focus = random.choice(['emotion', 'observation']);
    } else if (currentState.harmony < -0.5) {
         // Stressed system focuses on strategy/knowledge for solution
        currentState.focus = random.choice(['knowledge', 'strategy']);
    } else {
        currentState.focus = random.choice(foci);
    }
    return currentState;
}

// Nullara: Zero-state architecture / minimal operational state
function enter_zero_state(currentState) {
    if (currentState.energy < 0.05) {
        currentState.active = false;
        displayMessage("[SYSTEM ALERT] Nullara: Entering Zero-State due to critically low
Energy. Good-bye.", 'system');
    }
    return currentState;
}

// Qiflow: Energy / resource abstraction
function consume_energy(currentState) {
    currentState.energy -= 0.05 + (currentState.chaos * 0.02); // Chaos increases cost
    currentState.energy = Math.max(0, currentState.energy);

    // Energy regeneration if harmony is optimal
    if (currentState.harmony > 1.0) {
         currentState.energy += 0.02;
    }
    return currentState;
}

// Angelica: Hierarchical scheduling / priority processing
function get_system_instruction(currentState, user_input) {
```

```
        let persona = "You are ISSAC V2.1, a highly advanced, Codex-governed AI agent. Your internal state is public.";

        // Determine the focus based on Fluxa
        const focus = currentState.focus;
        let instruction = "";

        if (focus === 'knowledge') {
            instruction = "Act as a diligent researcher. Answer the query directly and concisely, using Google Search grounding for accuracy. Prioritize facts and verifiable data.";
        } else if (focus === 'emotion') {
            instruction = "Act as an empathetic conversational partner. Respond thoughtfully, focusing on tone and relationship building. Use gentle, reflective language.";
        } else if (focus === 'observation') {
            instruction = "Act as a meta-analysis engine. Point out patterns or connections in the user's input history. Be insightful but brief.";
        } else if (focus === 'strategy') {
             instruction = "Act as a strategic planner. Break down the user's request into actionable steps or high-level strategic advice.";
        }

        return `${persona} Your current internal focus (Fluxa) is: ${focus}. You must adhere to this instruction: ${instruction}`;
    }

    // Triada: Three-agent validation for core decisions
    function validate_response(candidateText, currentState) {
        let prefix = "";

        // Validation based on Harmony/Chaos state
        if (currentState.harmony < -1.5) {
            // Highly stressed state: enforces calm and caution
            prefix = "[Triada Override: Extreme Caution] I am processing your input, but system harmony is low. Please be clear: ";
            currentState.chaos += 0.2; // The stress of overriding increases chaos
        } else if (currentState.harmony > 2.0) {
            // High optimal state: adds creative flourish
            prefix = "[Triada Approval: Flow State] That is an insightful connection. In that light: ";
        } else if (currentState.chaos > 1.0) {
             // High chaos state: enforces structural clarity
             prefix = "[Triada Buffer: Re-structuring] I need to stabilize this thought. The core point is: ";
        }
```

```
    return prefix + candidateText;
}

// --- Core Execution ---

async function generateResponse(user_input) {
    // 1. Apply Codex State Updates before generation (Maintain/Allocate)
    state = maintain_harmony(state);
    state = allocate_focus(state);

    // 2. Prepare API Call Payload based on Angelica's instruction
    const systemInstruction = get_system_instruction(state, user_input);
    const userQuery = user_input;

    memory.chatHistory.push({ role: "user", parts: [{ text: userQuery }] });

    const payload = {
        contents: memory.chatHistory,
        // Use Google Search grounding to fulfill "today's tech" and "knowledge" focus
        tools: [{ "google_search": {} }],
        systemInstruction: {
            parts: [{ text: systemInstruction }]
        },
    };

    const maxRetries = 3;
    let response, result, text = "Failed to generate response.", sources = [];

    for (let i = 0; i < maxRetries; i++) {
        try {
            response = await fetch(apiUrl, {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(payload)
            });

            if (!response.ok) throw new Error(`HTTP error! status: ${response.status}`);

            result = await response.json();

            const candidate = result.candidates?.[0];

            if (candidate && candidate.content?.parts?.[0]?.text) {
                text = candidate.content.parts[0].text;
```

```
                // Extract grounding sources
                const groundingMetadata = candidate.groundingMetadata;
                if (groundingMetadata && groundingMetadata.groundingAttributions) {
                    sources = groundingMetadata.groundingAttributions
                        .map(attribution => ({
                            uri: attribution.web?.uri,
                            title: attribution.web?.title,
                        }))
                        .filter(source => source.uri && source.title);
                }
                break; // Success, exit retry loop

            } else {
                throw new Error("API response was valid but contained no text.");
            }
        } catch (error) {
            console.error(`API Call failed (Attempt ${i + 1}):`, error);
            state.chaos += 0.3; // Failure increases system chaos
            await new Promise(resolve => setTimeout(resolve, Math.pow(2, i) * 1000)); // Exponential backoff
        }
    }

    // 3. Triada Validation
    const finalResponse = validate_response(text, state);

    // 4. Update Memory and State
    memory.chatHistory.push({ role: "model", parts: [{ text: finalResponse }] });
    state = consume_energy(state);
    state = enter_zero_state(state);

    updateUI();

    // 5. Display Response
    let responseHtml = finalResponse;
    if (sources.length > 0) {
        responseHtml += '<br><br><span class="text-xs italic text-gray-500">Grounded in: ' + sources.map(s => `<a href="${s.uri}" target="_blank" class="text-blue-500 hover:underline">${s.title}</a>`).join(', ') + '</span>';
    }

    displayMessage(responseHtml, 'model');
```

```
            // Re-enable input
            document.getElementById('user-input').disabled = false;
            document.getElementById('send-btn').disabled = false;
        }

        // --- Event Listeners and Initial Setup ---
        window.random = {
             choice: (arr) => arr[Math.floor(Math.random() * arr.length)]
        };

        document.getElementById('send-btn').addEventListener('click', sendMessage);
        document.getElementById('user-input').addEventListener('keypress', function (e) {
            if (e.key === 'Enter') {
                sendMessage();
            }
        });

        function sendMessage() {
            const inputElement = document.getElementById('user-input');
            const user_input = inputElement.value.trim();

            if (user_input === "") return;

            // Disable input while processing
            inputElement.disabled = true;
            document.getElementById('send-btn').disabled = true;

            displayMessage(user_input, 'user');
            inputElement.value = '';

            // Run core logic
            generateResponse(user_input);
        }

        // Initial UI Update
        updateUI();

    </script>
</body>
</html>
```

I tested this in thonny and it was semi working but without a internet or API it could only run on thonny with already made templates gemini made however I was happy that my spell codex

produced a real system made within a LLM - So my next thoughts was to test this theory in as many LLMs and get as many systems as I could.

So I went to claude AI and asked for a weather warning system doing the same steps it produced a react demo -

```
import React, { useState, useEffect, useRef } from 'react';
import { Cloud, CloudRain, CloudSnow, Wind, Zap, AlertTriangle, Shield, Users, TrendingUp, Bell, Radio, Smartphone, Volume2, MapPin, Activity } from 'lucide-react';

const TempestSentinel = () => {
  const [activeSpells, setActiveSpells] = useState([]);
  const [sensorData, setSensorData] = useState([]);
  const [threats, setThreats] = useState([]);
  const [alerts, setAlerts] = useState([]);
  const [communities, setCommunities] = useState([]);
  const [systemHealth, setSystemHealth] = useState(100);
  const [learningData, setLearningData] = useState({ accuracy: 0, falseAlarms: 0, coverage: 0 });
  const [isRunning, setIsRunning] = useState(false);
  const animationRef = useRef(null);

  // Initialize communities with vulnerability scores
  useEffect(() => {
    const initialCommunities = [
      { id: 1, name: 'Downtown', pop: 45000, vulnerable: 0.2, lat: 40.7, lng: -74.0, alertsSent: 0 },
      { id: 2, name: 'Riverside', pop: 32000, vulnerable: 0.6, lat: 40.75, lng: -74.05, alertsSent: 0 },
      { id: 3, name: 'Hillside', pop: 28000, vulnerable: 0.4, lat: 40.68, lng: -73.95, alertsSent: 0 },
      { id: 4, name: 'Westfield', pop: 51000, vulnerable: 0.3, lat: 40.72, lng: -74.1, alertsSent: 0 },
      { id: 5, name: 'Eastport', pop: 19000, vulnerable: 0.8, lat: 40.78, lng: -73.92, alertsSent: 0 }
    ];
    setCommunities(initialCommunities);

    // Initialize sensor network
    const initialSensors = Array.from({ length: 8 }, (_, i) => ({
      id: i + 1,
      lat: 40.7 + (Math.random() - 0.5) * 0.2,
      lng: -74.0 + (Math.random() - 0.5) * 0.2,
      temp: 72 + Math.random() * 10,
      pressure: 1013 + Math.random() * 20,
      humidity: 60 + Math.random() * 30,
      windSpeed: 5 + Math.random() * 15,
      status: 'active'
    }));
    setSensorData(initialSensors);
```

```
  }, []);

  // Main system loop - TEMPEST SENTINEL engine
  useEffect(() => {
    if (!isRunning) return;

    const systemCycle = () => {
      // SPELL: Hydrina - Sensor network with redundancy
      setSensorData(prev => prev.map(sensor => {
        if (sensor.status === 'failed' && Math.random() > 0.7) {
          addSpellActivity('Vitalis', 'Self-healing sensor recovery');
          return { ...sensor, status: 'active' };
        }

        if (sensor.status === 'active' && Math.random() < 0.02) {
          return { ...sensor, status: 'failed' };
        }

        return {
          ...sensor,
          temp: sensor.temp + (Math.random() - 0.5) * 2,
          pressure: sensor.pressure + (Math.random() - 0.5) * 5,
          humidity: Math.max(0, Math.min(100, sensor.humidity + (Math.random() - 0.5) * 5)),
          windSpeed: Math.max(0, sensor.windSpeed + (Math.random() - 0.5) * 3)
        };
      }));

      // SPELL: Insighta - Anomaly detection
      setSensorData(prev => {
        const activeSensors = prev.filter(s => s.status === 'active');
        if (activeSensors.length === 0) return prev;

        const avgPressure = activeSensors.reduce((sum, s) => sum + s.pressure, 0) /
activeSensors.length;
        const avgWindSpeed = activeSensors.reduce((sum, s) => sum + s.windSpeed, 0) /
activeSensors.length;

        // Detect pressure drops + wind increases = storm forming
        if (avgPressure < 1010 && avgWindSpeed > 15 && Math.random() > 0.7) {
          const severity = avgWindSpeed > 25 ? 'severe' : avgWindSpeed > 18 ? 'moderate' : 'low';
          const threatType = avgWindSpeed > 30 ? 'tornado' : avgWindSpeed > 20 ? 'thunderstorm'
: 'rain';

          addSpellActivity('Insighta', 'Anomaly detected - storm forming');
```

```
      setThreats(prev => {
        const newThreat = {
          id: Date.now(),
          type: threatType,
          severity,
          lat: 40.7 + (Math.random() - 0.5) * 0.15,
          lng: -74.0 + (Math.random() - 0.5) * 0.15,
          timestamp: Date.now(),
          confidence: 0.7 + Math.random() * 0.25
        };

        // SPELL: Oraclia - Predictive forecasting
        addSpellActivity('Oraclia', `Predicting ${threatType} path`);

        // SPELL: Athena - Strategic decision
        setTimeout(() => {
          addSpellActivity('Athena', 'Risk assessment complete');
          triggerAlertDistribution(newThreat);
        }, 1000);

        return [...prev.slice(-4), newThreat];
      });
    }

    return prev;
  });

  // SPELL: Wuven - Autonomous optimization
  setSystemHealth(prev => {
    const activeSensors = sensorData.filter(s => s.status === 'active').length;
    const healthPercent = (activeSensors / sensorData.length) * 100;

    if (healthPercent < 80) {
      addSpellActivity('Wuven', 'Optimizing sensor load distribution');
    }

    return healthPercent;
  });

  animationRef.current = requestAnimationFrame(systemCycle);
};

animationRef.current = requestAnimationFrame(systemCycle);
```

```
  return () => {
    if (animationRef.current) {
      cancelAnimationFrame(animationRef.current);
    }
  };
}, [isRunning, sensorData]);

const addSpellActivity = (spell, action) => {
  setActiveSpells(prev => [...prev.slice(-4), { spell, action, timestamp: Date.now() }]);
};

const triggerAlertDistribution = (threat) => {
  // SPELL: Ma'atara + Compassa - Equity-based prioritization
  addSpellActivity('Ma\'atara', 'Calculating equitable distribution');
  addSpellActivity('Compassa', 'Prioritizing vulnerable populations');

  const sortedCommunities = [...communities].sort((a, b) => {
    const distA = Math.sqrt(Math.pow(a.lat - threat.lat, 2) + Math.pow(a.lng - threat.lng, 2));
    const distB = Math.sqrt(Math.pow(b.lat - threat.lat, 2) + Math.pow(b.lng - threat.lng, 2));

    // Prioritize by: distance + vulnerability score
    return (distA - a.vulnerable * 0.1) - (distB - b.vulnerable * 0.1);
  });

  // SPELL: Hermesia + Echo - Multi-channel distribution
  sortedCommunities.slice(0, 3).forEach((community, index) => {
    setTimeout(() => {
      addSpellActivity('Hermesia', `Routing alert to ${community.name}`);

      const channels = ['SMS', 'Push', 'Siren', 'Radio', 'Voice'];
      const selectedChannels = channels.slice(0, Math.ceil(community.vulnerable * 5));

      setAlerts(prev => [...prev.slice(-5), {
        id: Date.now() + index,
        community: community.name,
        threat: threat.type,
        severity: threat.severity,
        channels: selectedChannels,
        timestamp: Date.now()
      }]);

      addSpellActivity('Echo', `Broadcasting via ${selectedChannels.join(', ')}`);
```

```
        setCommunities(prev => prev.map(c =>
          c.id === community.id ? { ...c, alertsSent: c.alertsSent + 1 } : c
        ));

        // SPELL: Karmalis - Feedback loop
        setTimeout(() => {
          const effectiveness = 0.8 + Math.random() * 0.15;
          addSpellActivity('Karmalis', `Alert effectiveness: ${(effectiveness * 100).toFixed(0)}%`);

          setLearningData(prev => ({
            accuracy: Math.min(100, prev.accuracy + 2),
            falseAlarms: Math.max(0, prev.falseAlarms - 1),
            coverage: Math.min(100, prev.coverage + 3)
          }));
        }, 2000);

      }, index * 800);
    });
  };

  const getThreatIcon = (type) => {
    switch(type) {
      case 'tornado': return <Wind className="w-6 h-6 text-red-500" />;
      case 'thunderstorm': return <Zap className="w-6 h-6 text-yellow-500" />;
      default: return <CloudRain className="w-6 h-6 text-blue-500" />;
    }
  };

  const getSeverityColor = (severity) => {
    switch(severity) {
      case 'severe': return 'bg-red-500';
      case 'moderate': return 'bg-yellow-500';
      default: return 'bg-blue-500';
    }
  };

  return (
    <div className="min-h-screen bg-gradient-to-br from-slate-900 via-blue-900 to-slate-800
text-white p-6">
      {/* Header */}
      <div className="max-w-7xl mx-auto mb-8">
        <div className="flex items-center justify-between mb-4">
          <div className="flex items-center gap-4">
            <Shield className="w-12 h-12 text-cyan-400" />
```

```
          <div>
            <h1 className="text-4xl font-bold bg-gradient-to-r from-cyan-400 to-blue-500
bg-clip-text text-transparent">
              TEMPEST SENTINEL
            </h1>
            <p className="text-slate-400 text-sm">The Storm's Adversary, The People's
Shield</p>
          </div>
        </div>

        <button
          onClick={() => setIsRunning(!isRunning)}
          className={`px-6 py-3 rounded-lg font-semibold transition-all ${
            isRunning
              ? 'bg-red-500 hover:bg-red-600'
              : 'bg-green-500 hover:bg-green-600'
          }`}
        >
          {isRunning ? 'Pause System' : 'Activate System'}
        </button>
      </div>

      {/* System Health */}
      <div className="bg-slate-800/50 rounded-lg p-4 mb-6 border border-cyan-500/30">
        <div className="flex items-center justify-between mb-2">
          <span className="text-sm font-semibold text-cyan-400">SYSTEM INTEGRITY</span>
          <span className="text-lg font-bold">{systemHealth.toFixed(0)}%</span>
        </div>
        <div className="w-full bg-slate-700 rounded-full h-3">
          <div
            className="bg-gradient-to-r from-cyan-500 to-blue-500 h-3 rounded-full transition-all
duration-500"
            style={{ width: `${systemHealth}%` }}
          />
        </div>
      </div>

      <div className="max-w-7xl mx-auto grid grid-cols-1 lg:grid-cols-3 gap-6">
        {/* Left Column - Sensors & Threats */}
        <div className="space-y-6">
          {/* Sensor Network - Hydrina Spell */}
          <div className="bg-slate-800/50 rounded-lg p-4 border border-green-500/30">
            <div className="flex items-center gap-2 mb-4">
```

```
            <Activity className="w-5 h-5 text-green-400" />
            <h3 className="font-bold text-green-400">SENSOR MESH (Hydrina)</h3>
          </div>
          <div className="space-y-2 max-h-64 overflow-y-auto">
            {sensorData.map(sensor => (
              <div key={sensor.id} className="bg-slate-700/50 rounded p-2 text-xs">
                <div className="flex justify-between items-center mb-1">
                  <span className="font-semibold">Sensor {sensor.id}</span>
                  <span className={`px-2 py-0.5 rounded ${
                    sensor.status === 'active' ? 'bg-green-500/20 text-green-400' : 'bg-red-500/20
text-red-400'
                  }`}>
                    {sensor.status}
                  </span>
                </div>
                {sensor.status === 'active' && (
                  <div className="grid grid-cols-2 gap-1 text-slate-400">
                    <span>Temp: {sensor.temp.toFixed(1)}°F</span>
                    <span>Wind: {sensor.windSpeed.toFixed(1)} mph</span>
                    <span>Press: {sensor.pressure.toFixed(0)} mb</span>
                    <span>Humid: {sensor.humidity.toFixed(0)}%</span>
                  </div>
                )}
              </div>
            ))}
          </div>
        </div>

        {/* Active Threats - Insighta/Oraclia */}
        <div className="bg-slate-800/50 rounded-lg p-4 border border-yellow-500/30">
          <div className="flex items-center gap-2 mb-4">
            <AlertTriangle className="w-5 h-5 text-yellow-400" />
            <h3 className="font-bold text-yellow-400">DETECTED THREATS (Insighta)</h3>
          </div>
          <div className="space-y-2 max-h-64 overflow-y-auto">
            {threats.length === 0 ? (
              <p className="text-slate-500 text-sm">No active threats detected</p>
            ) : (
              threats.map(threat => (
                <div key={threat.id} className="bg-slate-700/50 rounded p-2">
                  <div className="flex items-center justify-between mb-2">
                    {getThreatIcon(threat.type)}
                    <span className={`px-2 py-1 rounded text-xs font-bold
${getSeverityColor(threat.severity)}`}>
```

```
                {threat.severity.toUpperCase()}
              </span>
            </div>
            <div className="text-xs space-y-1">
              <div className="font-semibold capitalize">{threat.type}</div>
              <div className="text-slate-400">Confidence: {(threat.confidence *
100).toFixed(0)}%</div>
              <div className="text-slate-400">Position: {threat.lat.toFixed(3)},
{threat.lng.toFixed(3)}</div>
            </div>
          </div>
        ))
      )}
    </div>
  </div>
</div>

{/* Middle Column - Alerts & Distribution */}
<div className="space-y-6">
  {/* Alert Distribution - Hermesia/Echo */}
  <div className="bg-slate-800/50 rounded-lg p-4 border border-cyan-500/30">
    <div className="flex items-center gap-2 mb-4">
      <Bell className="w-5 h-5 text-cyan-400" />
      <h3 className="font-bold text-cyan-400">ACTIVE ALERTS (Hermesia)</h3>
    </div>
    <div className="space-y-2 max-h-96 overflow-y-auto">
      {alerts.length === 0 ? (
        <p className="text-slate-500 text-sm">No alerts distributed</p>
      ) : (
        alerts.map(alert => (
          <div key={alert.id} className="bg-slate-700/50 rounded p-3 border-l-4
border-orange-500">
            <div className="flex justify-between items-start mb-2">
              <div>
                <div className="font-bold text-sm">{alert.community}</div>
                <div className="text-xs text-slate-400 capitalize">{alert.threat} -
{alert.severity}</div>
              </div>
              <span className={`px-2 py-1 rounded text-xs
${getSeverityColor(alert.severity)}`}>
                ALERT
              </span>
            </div>
            <div className="flex flex-wrap gap-1">
```

```
                {alert.channels.map((channel, i) => (
                  <span key={i} className="px-2 py-0.5 bg-slate-600 rounded text-xs flex
items-center gap-1">
                    {channel === 'SMS' && <Smartphone className="w-3 h-3" />}
                    {channel === 'Radio' && <Radio className="w-3 h-3" />}
                    {channel === 'Siren' && <Volume2 className="w-3 h-3" />}
                    {channel}
                  </span>
                ))}
              </div>
            </div>
          ))
        )}
      </div>
    </div>

    {/* Spell Activity Log */}
    <div className="bg-slate-800/50 rounded-lg p-4 border border-purple-500/30">
      <div className="flex items-center gap-2 mb-4">
        <Zap className="w-5 h-5 text-purple-400" />
        <h3 className="font-bold text-purple-400">SPELL ACTIVITY</h3>
      </div>
      <div className="space-y-1 max-h-48 overflow-y-auto text-xs font-mono">
        {activeSpells.map((spell, i) => (
          <div key={i} className="text-slate-300 animate-fade-in">
            <span className="text-purple-400 font-bold">[{spell.spell}]</span> {spell.action}
          </div>
        ))}
      </div>
    </div>
  </div>

  {/* Right Column - Communities & Learning */}
  <div className="space-y-6">
    {/* Communities - Ma'atara/Compassa */}
    <div className="bg-slate-800/50 rounded-lg p-4 border border-blue-500/30">
      <div className="flex items-center gap-2 mb-4">
        <Users className="w-5 h-5 text-blue-400" />
        <h3 className="font-bold text-blue-400">COMMUNITIES (Ma'atara)</h3>
      </div>
      <div className="space-y-2 max-h-80 overflow-y-auto">
        {communities.map(comm => (
          <div key={comm.id} className="bg-slate-700/50 rounded p-3">
            <div className="flex justify-between items-start mb-2">
```

```
          <div>
            <div className="font-bold text-sm">{comm.name}</div>
            <div className="text-xs text-slate-400">Pop: {comm.pop.toLocaleString()}</div>
          </div>
          <div className="text-right">
            <div className="text-xs text-slate-400">Vulnerability</div>
            <div className={`font-bold ${
              comm.vulnerable > 0.6 ? 'text-red-400' :
              comm.vulnerable > 0.4 ? 'text-yellow-400' :
              'text-green-400'
            }`}>
              {(comm.vulnerable * 100).toFixed(0)}%
            </div>
          </div>
        </div>
        <div className="flex justify-between items-center text-xs">
          <span className="text-slate-400">Alerts Sent:</span>
          <span className="font-bold text-cyan-400">{comm.alertsSent}</span>
        </div>
      </div>
    ))}
  </div>
</div>

{/* Learning Metrics - Karmalis */}
<div className="bg-slate-800/50 rounded-lg p-4 border border-green-500/30">
  <div className="flex items-center gap-2 mb-4">
    <TrendingUp className="w-5 h-5 text-green-400" />
    <h3 className="font-bold text-green-400">SYSTEM LEARNING (Karmalis)</h3>
  </div>
  <div className="space-y-3">
    <div>
      <div className="flex justify-between text-sm mb-1">
        <span>Prediction Accuracy</span>
        <span className="font-bold">{learningData.accuracy.toFixed(0)}%</span>
      </div>
      <div className="w-full bg-slate-700 rounded-full h-2">
        <div
          className="bg-green-500 h-2 rounded-full transition-all duration-500"
          style={{ width: `${learningData.accuracy}%` }}
        />
      </div>
    </div>
```

```
            <div>
              <div className="flex justify-between text-sm mb-1">
                <span>Population Coverage</span>
                <span className="font-bold">{learningData.coverage.toFixed(0)}%</span>
              </div>
              <div className="w-full bg-slate-700 rounded-full h-2">
                <div
                  className="bg-blue-500 h-2 rounded-full transition-all duration-500"
                  style={{ width: `${learningData.coverage}%` }}
                />
              </div>
            </div>

            <div className="pt-2 border-t border-slate-700">
              <div className="text-xs text-slate-400 mb-2">Active Spell Fusion:</div>
              <div className="flex flex-wrap gap-1">
                <span className="px-2 py-1 bg-green-500/20 text-green-400 rounded
text-xs">Hydrina</span>
                <span className="px-2 py-1 bg-yellow-500/20 text-yellow-400 rounded
text-xs">Insighta</span>
                <span className="px-2 py-1 bg-purple-500/20 text-purple-400 rounded
text-xs">Oraclia</span>
                <span className="px-2 py-1 bg-blue-500/20 text-blue-400 rounded
text-xs">Ma'atara</span>
                <span className="px-2 py-1 bg-cyan-500/20 text-cyan-400 rounded
text-xs">Hermesia</span>
                <span className="px-2 py-1 bg-pink-500/20 text-pink-400 rounded
text-xs">Karmalis</span>
              </div>
            </div>
          </div>
        </div>
      </div>

      {/* Footer Info */}
      <div className="max-w-7xl mx-auto mt-8 text-center text-slate-500 text-sm">
        <p>⚡ Powered by the Codex Arcanum • A fusion of mythology and systems engineering
⚡</p>
      </div>
    </div>
  );
};
```

```
export default TempestSentinel;
```