```tsx
import React, { useState, useCallback, useEffect } from 'react';
import {
  Send, Download, Copy, CheckCircle, TrendingUp,
  AlertCircle, Brain, Eye, EyeOff, Loader,
  Settings, ChevronRight, Share2, Activity
} from 'lucide-react';

/**
 * COMPOSANT SVG FLEUR DE LYS
 * Créé sur mesure pour l'esthétique Cognitive RAID
 */
const FleurDeLys = ({ className }: { className?: string }) => (
  <svg
    viewBox="0 0 24 24"
    fill="currentColor"
    className={className}
    xmlns="http://www.w3.org/2000/svg"
  >
    <path d="M12 2C12 2 8 6 8 9C8 11.5 9.5 13 11 13.5V16C9 16 6 14 6 11C6 9 5 8 4 8C3 8 2 9 2 11C2 15
5 17 8 18V21C6 21.5 5 22 5 22H19C19 22 18 21.5 16 21V18C19 17 22 15 22 11C22 9 21 8 20 8C19 8 18 9 18
11C18 14 15 16 13 16V13.5C14.5 13 16 11.5 16 9C16 6 12 2 12 2ZM12 6C13 6 13.5 8 13.5 9C13.5 10 13 11
12 11C11 11 10.5 10 10.5 9C10.5 8 11 6 12 6Z" />
  </svg>
);

export default function CognitiveRAID() {
  // ===== STATE MANAGEMENT =====
  const [step, setStep] = useState<'keys' | 'question' | 'processing' | 'results'>('keys');

  // Clés API pour les 5 moteurs du RAID
  const [apiKeys, setApiKeys] = useState({
    claude: '',
    chatgpt: '',
    gemini: '',
    grok: '',
    copilot: '' // Utilisera l'endpoint OpenAI avec un prompt système spécifique "Copilot"
  });

  const [showKeys, setShowKeys] = useState({
    claude: false,
    chatgpt: false,
    gemini: false,
    grok: false,
    copilot: false
  });

  const [question, setQuestion] = useState('');
  const [responses, setResponses] = useState<Record<string, string>>({});
  const [synthesis, setSynthesis] = useState<any>(null);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const [progress, setProgress] = useState('');

  // ===== MOTEUR CRAID (Logique V2.0 Optimisée) =====

  // 1. Extraction des Concepts (Stopwords Français)
  const extractConcepts = useCallback((text: string) => {
    if (!text || text.length < 50) return [];
    const words = text.match(/\b[a-zàâäéèêëïîôöùûüœæç]{5,}\b/gi) || [];
    const stopwords = new Set([
      'cette', 'comme', 'dans', 'pour', 'avec', 'sont', 'leurs', 'plus', 'peut',
      'être', 'fait', 'permet', 'avoir', 'faire', 'entre', 'donc', 'aussi', 'ainsi',
      'nous', 'vous', 'elles', 'notre', 'votre', 'cela', 'ceci', 'vers'
    ]);
```

```
    const freq: Record<string, number> = {};
    words.forEach(w => {
      const normalized = w.toLowerCase();
      if (!stopwords.has(normalized)) {
        freq[normalized] = (freq[normalized] || 0) + 1;
      }
    });
    return Object.entries(freq)
      .sort((a, b) => b[1] - a[1])
      .slice(0, 40)
      .map(([word]) => word);
  }, []);

  // 2. Segmentation
  const splitSentences = useCallback((text: string) => {
    if (!text) return [];
    return text
      .split(/[.!?]\s+|\n{2,}/)
      .map(s => s.trim())
      .filter(s => s.length > 20 && s.length < 500);
  }, []);

  // 3. Extraction des Affirmations (Claims)
  const extractClaims = useCallback((text: string) => {
    if (!text || text.length < 100) return [];
    const markers = [
      'est ', 'sont ', 'représente', 'correspond', 'signifie', 'implique',
      'démontre', 'prouve', 'est une', 'est le', 'est la', 'émerge', 'propose',
      'affirme', 'suggère', 'indique', 'montre', 'révèle', 'confirme', 'faut'
    ];
    const sentences = splitSentences(text);
    return sentences
      .filter(s => markers.some(m => s.toLowerCase().includes(m)))
      .slice(0, 25);
  }, [splitSentences]);

  // 4. Calcul de Similarité (Jaccard)
  const claimsSimilarity = useCallback((claim1: string, claim2: string) => {
    if (!claim1 || !claim2 || claim1.length < 10 || claim2.length < 10) return 0;
    const words1 = new Set(claim1.toLowerCase().match(/\b\w{4,}\b/g) || []);
    const words2 = new Set(claim2.toLowerCase().match(/\b\w{4,}\b/g) || []);
    if (words1.size === 0 || words2.size === 0) return 0;
    const intersection = [...words1].filter(w => words2.has(w)).length;
    const union = new Set([...words1, ...words2]).size;
    return union > 0 ? intersection / union : 0;
  }, []);

  // 5. Algorithme de Consensus (C)
  const findConsensus = useCallback((allResponses: Record<string, string>) => {
    const activeResponses = Object.entries(allResponses)
      .filter(([_, data]) => data && data.length > 100)
      .map(([key, content]) => ({ key, name: key.toUpperCase(), content }));

    if (activeResponses.length < 2) return { concepts: [], claims: [], confidence: 0 };

    // Concepts Communs
    const conceptSets = activeResponses.map(resp => new Set(extractConcepts(resp.content)));
    let commonConcepts = conceptSets.length > 0 ? Array.from(conceptSets[0]) : [];
    for (let i = 1; i < conceptSets.length; i++) {
      commonConcepts = commonConcepts.filter(c => conceptSets[i].has(c));
    }

    // Claims Communs
    const allClaims: any[] = [];
    activeResponses.forEach(resp => {
```

```
        extractClaims(resp.content).forEach(claim => {
          allClaims.push({ claim, ai: resp.name });
        });
      });

      const consensusClaims: any[] = [];
      const seen = new Set();

      // Limite pour éviter freeze UI
      const maxChecks = 600;
      let checks = 0;

      for (let i = 0; i < allClaims.length && checks < maxChecks; i++) {
        const claimNorm = allClaims[i].claim.toLowerCase().substring(0, 50);
        if (seen.has(claimNorm)) continue;
        seen.add(claimNorm);

        let supporters = [allClaims[i].ai];
        for (let j = i + 1; j < allClaims.length; j++) {
          checks++;
          if (claimsSimilarity(allClaims[i].claim, allClaims[j].claim) > 0.45) {
            if (!supporters.includes(allClaims[j].ai)) {
              supporters.push(allClaims[j].ai);
            }
          }
        }

        if (supporters.length >= 2) {
          consensusClaims.push({
            claim: allClaims[i].claim,
            support: supporters.length,
            ais: supporters,
            confidence: supporters.length / activeResponses.length
          });
        }
      }

      return {
        concepts: commonConcepts.slice(0, 20),
        claims: consensusClaims.sort((a, b) => b.confidence - a.confidence).slice(0, 10),
        confidence: commonConcepts.length > 0 ? Math.min(commonConcepts.length / 20, 1) : 0
      };
    }, [extractConcepts, extractClaims, claimsSimilarity]);

    // 6. Algorithme de Divergences (D) & Insights (I)
    const findDivergencesAndInsights = useCallback((allResponses: Record<string, string>) => {
      const activeResponses = Object.entries(allResponses)
        .filter(([_, content]) => content && content.length > 100)
        .map(([key, content]) => ({ key, name: key.toUpperCase(), content }));

      const divergences: any[] = [];
      const insights: Record<string, string[]> = {};

      if (activeResponses.length < 2) return { divergences, insights };

      activeResponses.forEach((resp, idx) => {
        // Divergences Conceptuelles
        const concepts = new Set(extractConcepts(resp.content));
        const otherConcepts = new Set();
        activeResponses.forEach((other, oidx) => {
          if (oidx !== idx) extractConcepts(other.content).forEach(c => otherConcepts.add(c));
        });
        const uniqueConcepts = [...concepts].filter(c => !otherConcepts.has(c)).slice(0, 8);
        if (uniqueConcepts.length > 0) {
          divergences.push({ ai: resp.name, concepts: uniqueConcepts });
```

```
      }

      // Insights Uniques (Claims)
      const claims = extractClaims(resp.content);
      const uniqueClaims: string[] = [];
      let checks = 0;

      claims.forEach(claim => {
        if (checks > 200) return;
        let isUnique = true;
        activeResponses.forEach((other, oidx) => {
          if (oidx === idx) return;
          extractClaims(other.content).forEach(otherClaim => {
            checks++;
            if (claimsSimilarity(claim, otherClaim) > 0.50) isUnique = false;
          });
        });
        if (isUnique) uniqueClaims.push(claim);
      });
      insights[resp.name] = uniqueClaims.slice(0, 4);
    });

    return { divergences, insights };
  }, [extractConcepts, extractClaims, claimsSimilarity]);

  // ===== API CALLS =====

  const callAPI = async (provider: string, q: string) => {
    try {
      if (provider === 'claude') {
        const res = await fetch('https://api.anthropic.com/v1/messages', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json', 'x-api-key': apiKeys.claude, 'anthropic-
version': '2023-06-01' },
          body: JSON.stringify({ model: 'claude-3-5-sonnet-20241022', max_tokens: 1500, messages: [{
role: 'user', content: q }] })
        });
        const data = await res.json();
        return data.content?.[0]?.text;
      }

      if (provider === 'chatgpt' || provider === 'copilot') {
        // Copilot simule un style plus "Microsoft/Corp" via prompt system
        const systemPrompt = provider === 'copilot'
          ? "You are Copilot, a helpful AI assistant. Be concise, professional, and focus on
practical application."
          : "You are a helpful assistant.";

        const res = await fetch('https://api.openai.com/v1/chat/completions', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json', 'Authorization': `Bearer ${provider ===
'copilot' ? apiKeys.copilot : apiKeys.chatgpt}` },
          body: JSON.stringify({
            model: 'gpt-4o', // ou gpt-4-turbo
            messages: [
              { role: 'system', content: systemPrompt },
              { role: 'user', content: q }
            ],
            max_tokens: 1500
          })
        });
        const data = await res.json();
        return data.choices?.[0]?.message?.content;
      }
```

```
      if (provider === 'gemini') {
        const res = await fetch(`https://generativelanguage.googleapis.com/v1beta/models/gemini-1.5-
flash:generateContent?key=${apiKeys.gemini}`, {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ contents: [{ parts: [{ text: q }] }] })
        });
        const data = await res.json();
        return data.candidates?.[0]?.content?.parts?.[0]?.text;
      }

      if (provider === 'grok') {
        // Grok utilise une interface compatible OpenAI
        const res = await fetch('https://api.x.ai/v1/chat/completions', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json', 'Authorization': `Bearer ${apiKeys.grok}` },
          body: JSON.stringify({
            model: 'grok-beta',
            messages: [{ role: 'user', content: q }],
            max_tokens: 1500
          })
        });
        const data = await res.json();
        return data.choices?.[0]?.message?.content;
      }

    } catch (e: any) {
      console.error(`${provider} Error:`, e);
      throw new Error(`${provider.toUpperCase()}: ${e.message || 'Error'}`);
    }
  };

  const handleLaunchRAID = async () => {
    if (!question.trim()) { setError('⚠️ Question requise'); return; }

    // Vérifier qu'au moins 2 clés sont présentes pour la redondance
    const activeKeys = Object.entries(apiKeys).filter(([_, k]) => k.length > 5);
    if (activeKeys.length < 2) { setError('⚠️ Le protocole RAID exige au moins 2 sources
d\'intelligence.'); return; }

    setLoading(true);
    setError(null);
    setResponses({});
    setStep('processing');

    const results: Record<string, string> = {};
    const providers = activeKeys.map(k => k[0]);

    // Exécution Parallèle (RAID 0 Striping simulé pour la vitesse)
    try {
      const promises = providers.map(async (provider) => {
        setProgress(`Connexion au noeud ${provider.toUpperCase()}...`);
        try {
          const text = await callAPI(provider, question);
          if (text) results[provider] = text;
        } catch (e: any) {
          console.error(e);
          // On continue même si un noeud échoue (Tolérance aux pannes RAID)
        }
      });

      await Promise.all(promises);

      if (Object.keys(results).length < 2) {
        throw new Error("Échec critique: Pas assez de réponses pour le consensus.");
```

```
      }

      setResponses(results);
      setProgress('Assemblage CRAID en cours...');

      // Synthèse locale
      await new Promise(r => setTimeout(r, 800)); // Petit délai pour UX

      const consensus = findConsensus(results);
      const { divergences, insights } = findDivergencesAndInsights(results);

      setSynthesis({
        question,
        timestamp: new Date().toISOString(),
        consensus,
        divergences,
        insights,
        sourceCount: Object.keys(results).length
      });

      setStep('results');

    } catch (err: any) {
      setError(err.message);
      setStep('question');
    } finally {
      setLoading(false);
      setProgress('');
    }
  };

  // ===== GENERATION MARKDOWN =====
  const generateMarkdown = () => {
    if (!synthesis) return '';
    const date = new Date(synthesis.timestamp).toLocaleString('fr-CA');
    let md = `# ⚜️ COGNITIVE RAID REPORT\n\n`;
    md += `**Date:** ${date}\n**Sources:** ${synthesis.sourceCount} Nodes\n**Question:**
${synthesis.question}\n\n---\n\n`;

    md += `## ✅ CONSENSUS (La Vérité Émergente)\n`;
    synthesis.consensus.claims.forEach((c: any) => {
      md += `- **[${Math.round(c.confidence * 100)}%]** ${c.claim}\n`;
    });
    md += `\n**Concepts Clés:** ${synthesis.consensus.concepts.join(', ')}\n\n`;

    md += `## 💡 INSIGHTS (Signaux Faibles)\n`;
    Object.entries(synthesis.insights).forEach(([ai, list]: any) => {
      if (list.length > 0) {
        md += `**${ai}:**\n${list.map((l: string) => `- ${l}`).join('\n')}\n`;
      }
    });

    md += `\n## 🔥 DIVERGENCES (Zones d'Incertitude)\n`;
    synthesis.divergences.forEach((d: any) => {
      md += `- **${d.ai}** focus sur: ${d.concepts.join(', ')}\n`;
    });

    md += `\n---\n*Généré par Cognitive RAID v1.0 | Bryan Ouellette*`;
    return md;
  };

  const copyToClipboard = () => {
    navigator.clipboard.writeText(generateMarkdown());
    alert('Rapport copié dans le presse-papier.');
```

```
  };

  // ===== RENDER =====
  return (
    <div className="min-h-screen text-white font-sans selection:bg-cyan-500 selection:text-black
overflow-x-hidden relative">

      {/* BACKGROUND FRACTAL / GRADIENT QUEBEC */}
      <div className="fixed inset-0 z-0 bg-[#000510]">
        <div className="absolute inset-0 bg-gradient-to-br from-black via-[#002060] to-black opacity-
90"></div>
        {/* Motif Fractal Subtil */}
        <div className="absolute inset-0 opacity-10" style={{
          backgroundImage: 'radial-gradient(circle at 2px 2px, rgba(255,255,255,0.15) 1px,
transparent 0)',
          backgroundSize: '32px 32px'
        }}></div>
      </div>

      <div className="relative z-10 max-w-5xl mx-auto px-6 py-12 flex flex-col min-h-screen">

        {/* HEADER */}
        <header className="flex flex-col items-center justify-center mb-16 space-y-4">
          <div className="flex items-center gap-6">
            <FleurDeLys className="w-12 h-12 text-white animate-pulse" />
            <h1 className="text-5xl md:text-7xl font-black tracking-tighter text-transparent bg-clip-
text bg-gradient-to-r from-white via-cyan-200 to-blue-400 drop-shadow-
[0_0_15px_rgba(0,100,255,0.5)]">
              COGNITIVE RAID
            </h1>
            <FleurDeLys className="w-12 h-12 text-white animate-pulse" />
          </div>
          <p className="text-blue-200 text-lg md:text-xl font-light tracking-widest uppercase border-
b border-blue-500/30 pb-2">
            Architecture de Symbiose Human-IA Distribuée
          </p>
        </header>

        {/* MAIN CONTENT AREA - GOLDEN RATIO PROPORTIONS */}
        <main className="flex-grow w-full">

          {/* STEP 1: API KEYS (THE VAULT) */}
          {step === 'keys' && (
            <div className="bg-black/40 backdrop-blur-xl border border-white/10 rounded-2xl p-8 md:p-
12 shadow-2xl animate-fade-in">
              <h2 className="text-3xl font-bold mb-8 flex items-center gap-3 text-cyan-400">
                <Settings className="w-8 h-8" /> Configuration du Cluster
              </h2>

              <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
                {[
                  { id: 'claude', label: 'Claude (Anthropic)', color: 'border-orange-500/50' },
                  { id: 'chatgpt', label: 'ChatGPT (OpenAI)', color: 'border-green-500/50' },
                  { id: 'gemini', label: 'Gemini (Google)', color: 'border-blue-500/50' },
                  { id: 'grok', label: 'Grok (xAI)', color: 'border-white/50' },
                  { id: 'copilot', label: 'Copilot (Microsoft)', color: 'border-pink-500/50' },
                ].map((provider) => (
                  <div key={provider.id} className={`bg-white/5 border ${provider.color} p-4 rounded-
xl transition hover:bg-white/10`}>
                    <label className="block text-sm font-bold mb-2 tracking-wide text-gray-300">
{provider.label}</label>
                    <div className="flex gap-2">
                      <input
                        type={showKeys[provider.id as keyof typeof showKeys] ? "text" : "password"}
                        value={apiKeys[provider.id as keyof typeof apiKeys]}
```

```
                        onChange={(e) => setApiKeys({...apiKeys, [provider.id]: e.target.value})}
                        className="w-full bg-black/50 border border-white/10 rounded px-3 py-2 text-
cyan-300 focus:outline-none focus:border-cyan-500 font-mono text-sm"
                        placeholder="sk-..."
                      />
                      <button
                        onClick={() => setShowKeys({...showKeys, [provider.id]: !showKeys[provider.id
as keyof typeof showKeys]})}
                        className="text-gray-400 hover:text-white"
                      >
                        {showKeys[provider.id as keyof typeof showKeys] ? <EyeOff size={18}/> : <Eye
size={18}/>}
                      </button>
                    </div>
                  </div>
                ))}
              </div>

              <div className="mt-10 flex justify-end">
                <button
                  onClick={() => setStep('question')}
                  className="group relative px-8 py-4 bg-white text-blue-900 font-black text-lg
rounded-full hover:shadow-[0_0_20px_rgba(255,255,255,0.4)] transition-all flex items-center gap-3"
                >
                  ACTIVER LE RAID <ChevronRight className="group-hover:translate-x-1 transition-
transform" />
                </button>
              </div>
            </div>
          )}

          {/* STEP 2: QUESTION (THE INPUT) */}
          {step === 'question' && (
            <div className="bg-black/40 backdrop-blur-xl border border-white/10 rounded-2xl p-8 md:p-
12 shadow-2xl animate-fade-in">
              <div className="flex justify-between items-center mb-6">
                <h2 className="text-3xl font-bold text-white">Requête Principale</h2>
                <button onClick={() => setStep('keys')} className="text-sm text-gray-400 hover:text-
white underline">Reconfigurer Clés</button>
              </div>

              <textarea
                value={question}
                onChange={(e) => setQuestion(e.target.value)}
                placeholder="Entrez votre question complexe ici. Le RAID va la distribuer et
synthétiser la vérité émergente..."
                className="w-full h-64 bg-black/30 border border-blue-500/30 rounded-xl p-6 text-xl
text-white placeholder-blue-300/30 focus:outline-none focus:border-cyan-400 focus:ring-1 focus:ring-
cyan-400 transition-all resize-none font-light leading-relaxed"
              />

              {error && (
                <div className="mt-4 p-4 bg-red-900/20 border border-red-500 text-red-300 rounded
flex items-center gap-3">
                  <AlertCircle /> {error}
                </div>
              )}

              <div className="mt-8 flex justify-end">
                <button
                  onClick={handleLaunchRAID}
                  disabled={loading}
                  className="px-10 py-5 bg-gradient-to-r from-blue-600 to-cyan-600 text-white font-
black text-xl rounded-xl hover:scale-105 hover:shadow-[0_0_30px_rgba(0,100,255,0.6)] transition-all
disabled:opacity-50 disabled:scale-100 flex items-center gap-4"
```

```
                >
                  <Brain className="w-8 h-8" /> LANCER LA SYNTHÈSE
                </button>
              </div>
            </div>
          )}

          {/* STEP 3: PROCESSING (THE CRUNCH) */}
          {step === 'processing' && (
            <div className="flex flex-col items-center justify-center h-96 animate-fade-in">
              <div className="relative">
                <div className="w-32 h-32 border-4 border-blue-900 rounded-full animate-spin border-
t-cyan-400"></div>
                <FleurDeLys className="absolute top-1/2 left-1/2 transform -translate-x-1/2 -
translate-y-1/2 w-12 h-12 text-white animate-pulse" />
              </div>
              <h3 className="mt-8 text-2xl font-bold text-cyan-300 animate-pulse">{progress}</h3>
              <p className="text-blue-300 mt-2">Réduction de l'entropie en cours...</p>
            </div>
          )}

          {/* STEP 4: RESULTS (THE TRUTH) */}
          {step === 'results' && synthesis && (
            <div className="space-y-8 animate-fade-in">

              {/* STATUS BAR */}
              <div className="flex items-center justify-between bg-white/5 rounded-full px-6 py-3
border border-white/10">
                <div className="flex items-center gap-4">
                  <span className="flex h-3 w-3 relative">
                    <span className="animate-ping absolute inline-flex h-full w-full rounded-full bg-
green-400 opacity-75"></span>
                    <span className="relative inline-flex rounded-full h-3 w-3 bg-green-500"></span>
                  </span>
                  <span className="text-sm font-mono text-cyan-300">RAID COMPLETED
({synthesis.sourceCount} NODES)</span>
                </div>
                <div className="flex gap-2">
                  <button onClick={copyToClipboard} className="p-2 hover:bg-white/10 rounded-full
text-white"><Copy size={20}/></button>
                  <button onClick={() => { setStep('question'); setSynthesis(null); }} className="p-2
hover:bg-white/10 rounded-full text-white"><Share2 size={20}/></button>
                </div>
              </div>

              {/* CONSENSUS CARD (HERO) */}
              <div className="bg-gradient-to-br from-blue-900/40 to-black border border-cyan-500/50
rounded-2xl p-8 md:p-10 shadow-[0_0_40px_rgba(0,50,150,0.3)]">
                <h2 className="text-3xl font-bold text-white mb-6 flex items-center gap-3">
                  <CheckCircle className="text-green-400 w-8 h-8" /> CONSENSUS
                </h2>
                {synthesis.consensus.claims.length > 0 ? (
                  <ul className="space-y-6">
                    {synthesis.consensus.claims.map((claim: any, i: number) => (
                      <li key={i} className="flex gap-4 items-start group">
                        <div className="mt-1 min-w-[3rem] text-right font-mono text-green-400 font-
bold border-r border-white/20 pr-3">
                          {Math.round(claim.confidence * 100)}%
                        </div>
                        <p className="text-gray-100 text-lg leading-relaxed group-hover:text-white
transition-colors">
                          {claim.claim}
                        </p>
                      </li>
                    ))}
```

```
                  </ul>
                ) : (
                  <p className="text-gray-400 italic">Aucun consensus fort détecté. Le sujet est
hautement controversé.</p>
                )}

                {synthesis.consensus.concepts.length > 0 && (
                  <div className="mt-8 pt-6 border-t border-white/10 flex flex-wrap gap-2">
                    {synthesis.consensus.concepts.map((c: string, i: number) => (
                      <span key={i} className="px-3 py-1 bg-cyan-900/30 text-cyan-300 text-xs font-
mono uppercase tracking-wider rounded border border-cyan-500/20">
                        {c}
                      </span>
                    ))}
                  </div>
                )}
              </div>

              <div className="grid grid-cols-1 md:grid-cols-2 gap-8">
                {/* INSIGHTS CARD */}
                <div className="bg-black/40 border border-purple-500/30 rounded-2xl p-8">
                  <h2 className="text-2xl font-bold text-purple-300 mb-6 flex items-center gap-3">
                    <TrendingUp className="w-6 h-6" /> INSIGHTS UNIQUES
                  </h2>
                  <div className="space-y-6">
                    {Object.entries(synthesis.insights).map(([ai, list]: any) => list.length > 0 && (
                      <div key={ai}>
                        <h3 className="text-xs font-bold text-gray-500 uppercase tracking-widest mb-
2">{ai}</h3>
                        <ul className="space-y-2 border-l-2 border-purple-500/30 pl-4">
                          {list.map((l: string, i: number) => (
                            <li key={i} className="text-sm text-gray-300">{l}</li>
                          ))}
                        </ul>
                      </div>
                    ))}
                  </div>
                </div>

                {/* DIVERGENCES CARD */}
                <div className="bg-black/40 border border-orange-500/30 rounded-2xl p-8">
                  <h2 className="text-2xl font-bold text-orange-300 mb-6 flex items-center gap-3">
                    <AlertCircle className="w-6 h-6" /> DIVERGENCES
                  </h2>
                  <div className="space-y-4">
                    {synthesis.divergences.length > 0 ? synthesis.divergences.map((div: any, i:
number) => (
                      <div key={i} className="bg-orange-900/10 p-4 rounded border border-orange-
500/20">
                        <div className="font-bold text-orange-200 mb-2">{div.ai}</div>
                        <div className="flex flex-wrap gap-2">
                          {div.concepts.map((c: string, j: number) => (
                            <span key={j} className="text-xs text-orange-400 bg-orange-900/30 px-2
py-1 rounded">{c}</span>
                          ))}
                        </div>
                      </div>
                    )) : (
                      <p className="text-gray-500 text-sm">Alignement parfait entre les modèles.</p>
                    )}
                  </div>
                </div>
              </div>

              {/* ACTION BUTTON */}
```

```
              <div className="flex justify-center mt-12 pb-12">
                <button
                  onClick={() => { setStep('question'); setSynthesis(null); setQuestion(''); }}
                  className="px-8 py-3 bg-white/5 hover:bg-white/10 border border-white/20 rounded-
full text-white transition flex items-center gap-2"
                >
                  <Activity size={18} /> NOUVELLE ANALYSE
                </button>
              </div>

            </div>
          )}

        </main>

        {/* FOOTER */}
        <footer className="mt-auto py-8 text-center border-t border-blue-900/30">
          <div className="flex justify-center items-center gap-2 mb-2">
            <FleurDeLys className="w-4 h-4 text-blue-500" />
            <span className="text-blue-500 font-mono text-sm tracking-widest">COGNITIVE RAID
v1.0</span>
            <FleurDeLys className="w-4 h-4 text-blue-500" />
          </div>
          <p className="text-gray-500 text-xs">
            Powered by CRAID & Bryan Ouellette. Minimizing Cognitive Entropy.
          </p>
        </footer>

      </div>
    </div>
  );
}
```