C'est une excellente idée. En renvoyant les résultats dans CaSS, vous transformez votre script d'un simple outil de calcul en un **véritable service de synchronisation d'état**.

Voici la version augmentée du script. J'ai ajouté une méthode pushResultsToCass qui crée (ou met à jour) un objet de type **"CeredisProfile"** pour chaque apprenant. Cela permettra à n'importe quel tableau de bord ou interface front-end de lire le niveau calculé directement depuis l'API CaSS.

### Script Node.js avec synchronisation API (POST)

JavaScript

const axios = require('axios');  
  
// --- CONFIGURATION CEREDIS ---  
const CONFIG = {  
 baseUrl: process.env.CASS\_URL || 'https://cass.ceredis.net',  
 apiKey: process.env.CASS\_API\_KEY, // Optionnel si votre instance nécessite une clé  
 weights: {  
 P1: 0.15, P2: 0.30, P3: 0.35, P4: 0.20  
 },  
 domains: {  
 D1: { weight: 0.20, compPrefix: "1." },  
 D2: { weight: 0.20, compPrefix: "2." },  
 D3: { weight: 0.25, compPrefix: "3." },  
 D4: { weight: 0.15, compPrefix: "4." },  
 D5: { weight: 0.20, compPrefix: "5." }  
 }  
};  
  
class CeredisEngine {  
  
 // 1. Récupération des assertions  
 async fetchAssertions(agentId) {  
 const url = `${CONFIG.baseUrl}/api/data?q=assertion&subject=${agentId}`;  
 const response = await axios.get(url);  
 return response.data;  
 }  
  
 // 2. Calcul du Score de Compétence  
 calculateCompScore(assertions) {  
 if (!assertions.length) return 0;  
 let total = 0, weightSum = 0;  
 assertions.forEach(a => {  
 const w = CONFIG.weights[a.ceredis?.evidenceType];  
 if (w && a.ceredis?.rawScore) {  
 total += (a.ceredis.rawScore \* w);  
 weightSum += w;  
 }  
 });  
 return weightSum > 0 ? (total / weightSum) : 0;  
 }  
  
 // 3. Dérivation du niveau CECRL  
 deriveCECRL(score, domainScores, hasP4) {  
 let level = "A1";  
 if (score >= 500) level = "C1";  
 else if (score >= 400) level = "B2";  
 else if (score >= 300) level = "B1";  
 else if (score >= 200) level = "A2";  
  
 // Règle restrictive : B2/C1 impossible sans preuve P4  
 if ((level === "B2" || level === "C1") && !hasP4) {  
 const lower = { "C1": "B2", "B2": "B1" };  
 return lower[level] + "+ (Transition)";  
 }  
 return level;  
 }  
  
 // 4. PUSH vers CaSS (La nouveauté)  
 async pushResultsToCass(result) {  
 // Nous créons un objet "CreativeWork" personnalisé pour stocker le profil calculé  
 const profileObject = {  
 "@context": "https://schema.cassproject.org/0.4",  
 "@type": "CreativeWork",  
 "name": `CEREDIS Profile - ${result.agentId}`,  
 "publisher": result.agentId,  
 "ceredisData": {  
 "globalScore": result.ceredisScore,  
 "cecrlLevel": result.cecrlLevel,  
 "domainBreakdown": result.domainScores,  
 "lastUpdated": result.timestamp  
 }  
 };  
  
 try {  
 // POST vers l'endpoint data de CaSS  
 await axios.post(`${CONFIG.baseUrl}/api/data`, profileObject);  
 console.log(`✅ Profil mis à jour dans CaSS pour : ${result.agentId}`);  
 } catch (error) {  
 console.error(`❌ Erreur lors du POST vers CaSS : ${error.message}`);  
 }  
 }  
  
 // 5. Orchestration  
 async runForAgent(agentId) {  
 const assertions = await this.fetchAssertions(agentId);  
 let domainAgg = { D1: [], D2: [], D3: [], D4: [], D5: [] };  
 let hasP4 = false;  
  
 assertions.forEach(ass => {  
 if (ass.ceredis?.evidenceType === 'P4') hasP4 = true;  
 Object.keys(CONFIG.domains).forEach(d => {  
 if (ass.competency.includes(CONFIG.domains[d].compPrefix)) domainAgg[d].push(ass);  
 });  
 });  
  
 let domainScores = {};  
 let weightedSum = 0;  
 for (const d in domainAgg) {  
 domainScores[d] = this.calculateCompScore(domainAgg[d]);  
 weightedSum += (domainScores[d] \* CONFIG.domains[d].weight);  
 }  
  
 const finalResult = {  
 agentId,  
 ceredisScore: (weightedSum \* 6).toFixed(2),  
 domainScores,  
 cecrlLevel: this.deriveCECRL(weightedSum \* 6, domainScores, hasP4),  
 timestamp: new Date().toISOString()  
 };  
  
 // Envoi automatique des résultats vers CaSS  
 await this.pushResultsToCass(finalResult);  
 return finalResult;  
 }  
}  
  
// --- INITIALISATION ---  
const engine = new CeredisEngine();  
// Remplacez par une boucle sur vos IDs utilisateurs réels  
engine.runForAgent("user\_identifier\_123");

### Ce que ce script change pour vous :

1. **Rémanence des données** : Le niveau CECRL n'est plus seulement affiché dans une console, il devient un objet interrogeable dans votre base de données CaSS (https://cass.ceredis.net/api/data?q=CEREDIS Profile).
2. **Calcul "Trigger"** : Vous pouvez configurer ce script pour qu'il s'exécute dès qu'une nouvelle preuve est ajoutée, garantissant que le niveau affiché à l'élève est toujours à jour.
3. **Auditabilité** : En enregistrant l'objet ceredisData, vous gardez une trace historique des scores au moment du calcul, ce qui est crucial pour les rapports de recherche.

### Comment l'installer sur Coolify ?

* Créez une **Node.js Application**.
* Collez ce code dans votre index.js.
* Dans les paramètres de déploiement, vous pouvez utiliser un **Cron Job** intégré à Coolify pour lancer le script toutes les X minutes (ex: \*/15 \* \* \* \* pour toutes les 15 minutes).

**Souhaitez-vous que je vous aide à structurer le tableau de bord (Front-end) qui lira ces informations pour les afficher aux apprenants ?**