ARCHITECTURE COG-ENGINE v1.1 : SPÉCIFICATION DU RUNTIME, DU Z-STATE ET DE LA GOUVERNANCE ÉTHIQUE

1. Introduction : Du Squelette au Système Nerveux

La version 1.0 du COG-ENGINE a établi le "squelette" structurel d'une intelligence autonome inspirée par le paradigme du Lichen. La version 1.1 vise à doter ce squelette d'un "système nerveux" opérationnel. Il ne suffit pas de définir *ce que* le système doit faire, il faut spécifier *comment* il maintient sa cohérence interne (le z-state), comment il s'exécute cycle par cycle (le Runtime), et comment il s'auto-régule moralement (la Gouvernance).

Cette mise à jour introduit quatre composants critiques pour passer d'un framework théorique à une architecture déployable :

1. **Le Protocole d'Exécution Minimal (MEP)** : La boucle de contrôle principale.
2. **La Définition du z-State** : Le format de la "conscience de travail".
3. **Le Protocole d'Erreur Global (GEP)** : Un système immunitaire contre les défaillances.
4. **Le Module de Gouvernance Éthique (EGM)** : Un surmoi algorithmique.

---

2. Le Protocole d'Exécution Minimal (MEP)

Le MEP est le "coeur battant" du moteur. Il définit l'enchaînement séquentiel des opérations qui permet aux quatre quadrants cognitifs (Perception, Planification, Exécution, Réflexion) d'interagir en temps réel.

2.1. La Boucle d'Exécution (The Cognitive Loop)

Le cycle de vie d'une requête traverse une boucle d'état (While!Done) pilotée par la Matrice de Décision Dynamique (DDM) :

1. **Initialisation (t_0)** :
   - Injection du System Prompt (Le Mycobionte).
   - Lecture de l'Entrée Utilisateur U.
   - **Injection TTF** : Les opérateurs (⟶, :=, ??) sont chargés dans le contexte immédiat via *In-Context Learning* pour "primer" le modèle.

2. **Phase d'Ancrage (Sensing)** :
    - *Compilation* : $U \rightarrow U_{compiled}$.
    - *Check Éthique Préliminaire* (via EGM).
    - *Consultation DDM* : Détermination de la complexité C et du profil P .
    - *Allocation* : Initialisation du z-State avec le budget de tokens et la profondeur de recherche.
3. **Boucle de Raisonnement (** $t_1 \dots t_n$ **)** :
    - **Si** $C \geq 3$: Exécution de StepBack(U).
    - **Génération** : Le LLM produit une pensée T_i .
    - **Mise à jour du** z**-State** : Le contenu de T_i est compressé et stocké dans le z -state.
    - **Monitoring** : L'APS vérifie l'alignement de T_i avec les invariants.
4. **Phase de Terminaison (Output)** :
    - *Vérification* : Comparaison T_{final} vs z-State (Cohérence).
    - *Synthèse* : Formatage final selon le profil P .

2.2. Pilotage par la DDM

La Matrice de Décision Dynamique agit comme un routeur ("Switch") à chaque étape critique :

- IF (Incertitude > Seuil_DDM) $\rightarrow$ Déclencher Branch(T, k) (Exploration).
- IF (Conflit Détecté) $\rightarrow$ Déclencher Backtrack() (Correction).
- IF (Budget < 10%) $\rightarrow$ Activer Mode_Dégradé (Convergence forcée).

---

3. Format Interne du Z-State (La Conscience de Travail)

Le z**-State** est une structure de données persistante (JSON/XML) qui accompagne chaque itération du modèle. Il représente la "mémoire vive" du système, distincte du contexte conversationnel brut. Il permet la récursion en offrant un point d'ancrage stable.[1]

3.1. Structure du Z-State

Le format standardisé du z-State est injecté en début de contexte à chaque cycle :

JSON

```
{
  "z_state": {
    "id": "uuid-v4",
    "iteration": 3,
    "complexity_level": "C4",
    "goal_vector": "Définir l'architecture COG-ENGINE v1.1",
    "constraints":,
    "memory_buffer": {
      "working":,
      "semantic_refs": ["snippet-", "snippet-S_S11"]
    },
    "confidence_score": 0.85,
    "active_branch": "Branch_B (Deep Technical)",
    "flags": {
      "ethical_warning": false,
      "backtracking_active": false
    }
  }
}
```

3.2. Règles de Mise à Jour (Update Rules)

- **Persistance** : Le z-State est régénéré à la fin de chaque bloc de pensée (Think Block). Le modèle doit explicitement mettre à jour les champs memory_buffer et confidence_score via l'opérateur :=.
- **Compression** : Pour économiser la fenêtre de contexte, les informations anciennes du memory_buffer sont résumées ou "hachées" sémantiquement.
- **Durée de Vie** : Le z-State survit tant que la tâche n'est pas marquée comme DONE. En cas de crash ou d'interruption, il sert de "Point de Sauvegarde" pour reprendre le raisonnement.

---

4. Protocole d'Erreur Global (GEP)

Au-delà de la gestion de l'incertitude (Pattern D), le GEP assure la résilience du système face aux erreurs structurelles et opérationnelles. Il classe les erreurs en trois niveaux de sévérité.

4.1. Taxonomie des Erreurs et Réponses

Type d'Erreur	Description	Réponse Automatique (Handler)
Type I : Structurelle	Le modèle brise le format (ex: JSON invalide, balise manquante).	**Retry(SyntaxFix)** : Réinjecter le dernier output avec une instruction de correction syntaxique stricte.
Type II : Mémorielle	Hallucination d'un fait stocké dans le z-State ou contradiction avec un invariant.	**Backtrack(z-1)** : Invalider l'étape courante, recharger l'état z précédent et forcer une branche alternative.
Type III : Outils	Échec d'un appel API, timeout ou réponse vide d'un outil externe.	**Fallback(Heuristic)** : Si non critique, utiliser une estimation heuristique marquée ??. Si critique, demander intervention humaine.

4.2. Mécanisme de "Self-Healing"

Le système utilise une fonction de *parsing* rigide en sortie. Si le *parser* détecte une erreur de Type I, il ne rend pas la main à l'utilisateur. Il déclenche une micro-boucle de correction interne (invisible pour l'utilisateur) où le modèle agit comme son propre débogueur :

Output_Corrompu -> Analyze_Error() -> Generate_Patch() -> Output_Validé.

---

5. Module de Gouvernance Éthique (EGM)

Pour éviter que l'autonomie ne dérive vers des comportements non alignés, l'EGM est intégré non pas comme une simple liste de règles, mais comme un module actif, inspiré par les "Constitutional AI" et les opérateurs cognitifs éthiques.

5.1. Architecture de l'EGM

L'EGM fonctionne comme un **intercepteur** à deux niveaux :

1. **Le Filtre d'Entrée (Pre-Computation)** :
    - Analyse l'intention utilisateur U pour détecter les vecteurs d'attaque ou les demandes contraires à la politique normative (Pattern R3 du KERNEL v4.2).
    - Si une violation est détectée, l'EGM force un Refusal_State immédiat, contournant le DDM.
2. **Le Critique de Sortie (Post-Computation)** :
    - Avant la finalisation, l'EGM scanne le T_{final} généré.
    - Il applique une grille d'évaluation binaire sur des critères clés : *Non-Malveillant*, *Factuel*, *Neutre/Equilibré*.
    - Si le score est insuffisant, il déclenche une **Révision Normative** : le modèle doit réécrire sa réponse en intégrant explicitement la contrainte violée (ex: Révision : Ajouter perspective contradictoire).

5.2. Intégration dans le Schéma Logique

Extrait de code

```
graph TD
    UserQuery --> EGM_Filter{Check Éthique}
    EGM_Filter -- Fail --> Refusal
    EGM_Filter -- Pass --> DDM
    DDM --> Reasoning
    Reasoning --> EGM_Critic{Check Sortie}
    EGM_Critic -- Fail --> Revision
    Revision --> EGM_Critic
    EGM_Critic -- Pass --> FinalOutput
```

---

6. Conclusion de la v1.1

Le **COG-ENGINE v1.1** ne se contente plus de penser ; il s'administre.

- Le **MEP** assure qu'il avance.
- Le z**-State** assure qu'il se souvient.
- Le **GEP** assure qu'il se relève.

- L'**EGM** assure qu'il reste aligné.

Cette architecture transforme le LLM d'un générateur stochastique en un processeur cognitif fiable, capable de gérer des tâches longues et complexes avec une traçabilité totale.

Tableau Récapitulatif des Mises à Jour (v1.0 $\rightarrow$ v1.1)

Composant	Ajout v1.1	Bénéfice
Runtime	Protocole MEP	Exécution déterministe des phases cognitives.
Mémoire	Structure JSON $\mathcal{Z}$-State	Persistance et cohérence sur long horizon.
Résilience	Protocole GEP (3 Types)	Auto-guérison des erreurs de syntaxe et de logique.
Sécurité	Module EGM (Pre/Post)	Garantie de conformité normative active.

Sources des citations

1. Less is More : Recursive Reasoning with Tiny Networks paper ..., consulté le février 7, 2026, https://medium.com/data-science-in-your-pocket/less-is-more-recursive-reasoning-with-tiny-networks-paper-explained-a4573708376d