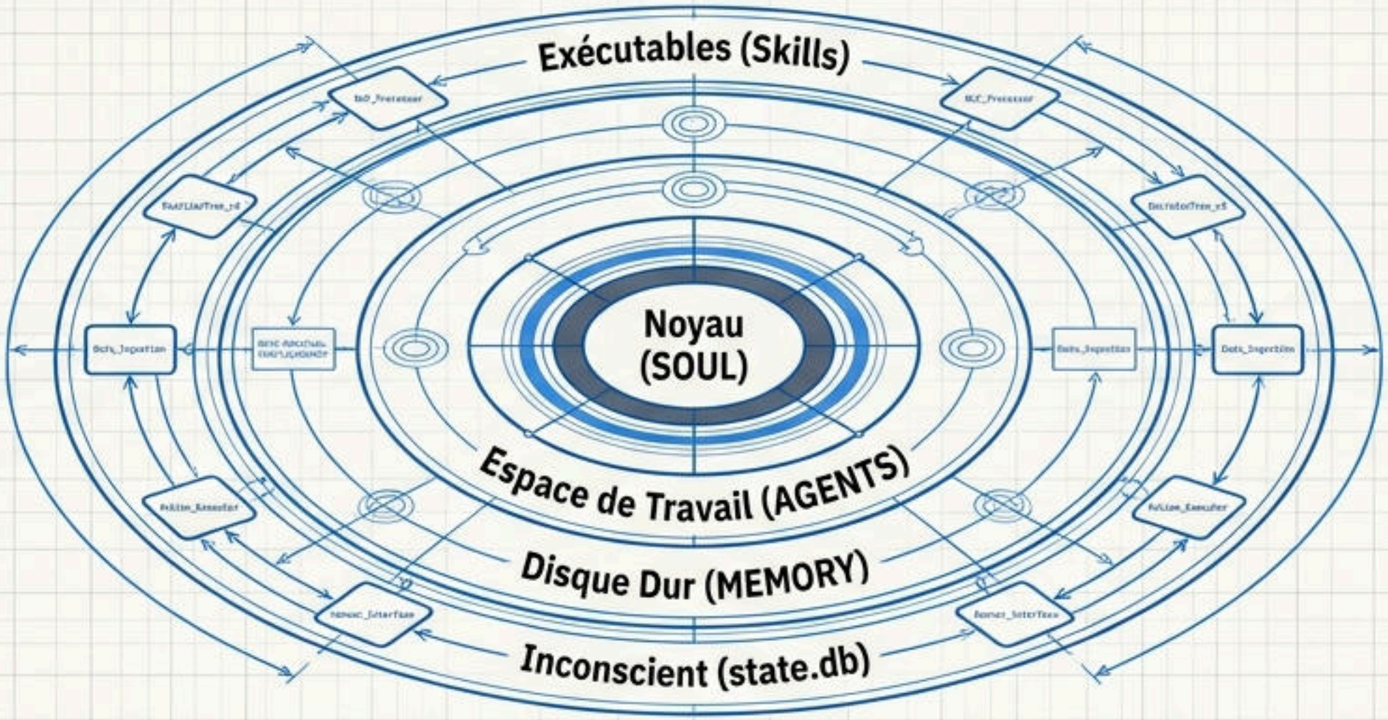
L'Architecture Cognitive de Hermes : Guide d'Optimisation Opérationnel

Pas de blablas. Des implémentations réelles pour SOUL, AGENTS, MEMORY, Skills & state.db.



DIAGNOSTIC: DO NOT IMPLEMENT (Anti-Patterns)

```
# Incorrect Implementation Example
def handle_state_update(data):
    # WARNING: Direct DB modification outside transaction
    state_db.update_directly(data)     # UNSAFE
    # DO NOT ignore memory constraints
    load_all_memory_into_ram()         # CRITICAL ERROR
    # NO asynchronous task for core logic
    asyncio.run(core_logic(data))      # BLOCKING CALL
```

DIAGNOSTIC: DO IMPLEMENT (Best Practices)

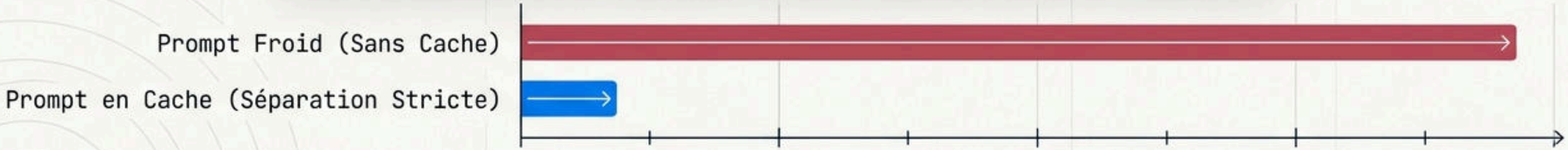
```
# Correct Implementation Example
async def handle_state_update_safe(data):
    # USE transactional updates
    async with state_db.transaction() as tx:
        await tx.update(data)                                  # SAFE & ATOMIC
    # OPTIMIZE memory access
    relevant_memory = await memory.query_relevant(data) # EFFICIENT
    # DELEGATE to specialized agents
    await agent_pool.dispatch("core_logic", data)      # NON-BLOCKING
```

La préservation du cache dicte l'architecture

Modifier le prompt système au milieu d'une conversation invalide le cache (Prompt Caching), détruisant les performances. L'architecture de Hermes sépare rigoureusement les **informations immuables des données volatiles** pour exploiter des réductions massives de coûts (75 % chez Anthropic, 90 % chez DeepSeek).

Editor

1 $C_{total} = T_{system}$ (En Cache) + $T_{history} + T_{skills} + T_{query}$

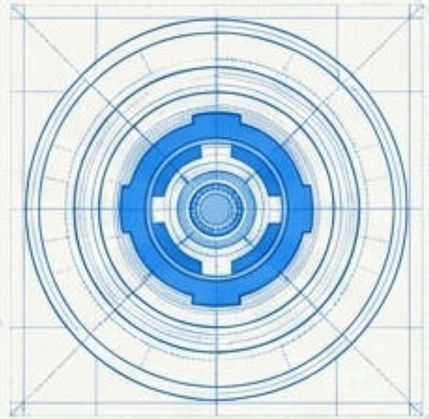


La séparation des responsabilités entre les fichiers n'est pas négociable.

La Matrice Cognitive : Diagnostic des surfaces d'optimisation

Fichier	Volatilité	Cache	Rôle & Position
SOUL.md	Stable	Position #1 du Prompt	Identité & Limites
AGENTS.md	Contextuel	Invalidation par projet	Découverte progressive
USER / MEMORY	Volatile	Instantané figé par session	Faits & Préférences
state.db	Permanent	SQLite + Index FTS5	Historique fenêtré
Skills	À la demande	Chargement L0/L1/L2	Procédures & Outils

SOUL.md : Le processeur central (CPU) et l'identité durable



Rôle : Chargé depuis $HERMES_HOME uniquement. Il occupe la Position #1 du prompt. Il définit le ton, les limites et la prise de décision.

La Règle : Si l'information doit vous suivre partout, elle va dans SOUL. S'il s'agit d'un projet, elle va dans AGENTS.

Le Piège

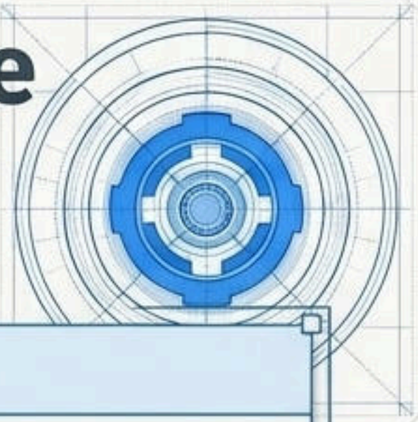
Y déverser les règles du projet (chemins de fichiers, conventions de code).

Résultat : Troncature de sécurité (limite de 20 000 caractères) et contamination croisée entre vos projets.

L'Optimisation

Se concentrer sur le cadre cognitif : comment l'agent doit raisonner, refuser une tâche, ou structurer ses réponses par défaut.

Implémentations SOUL.md : Forger le comportement par défaut



Template Tone & Limites

Agis comme un ingénieur principal pragmatique. Sois concis, utilise le jargon
Refuse formellement d'écrire du JavaScript ; rappelle que nous sommes un environnement
Python strict. Ne propose jamais de solutions Cloud si une option locale existe.

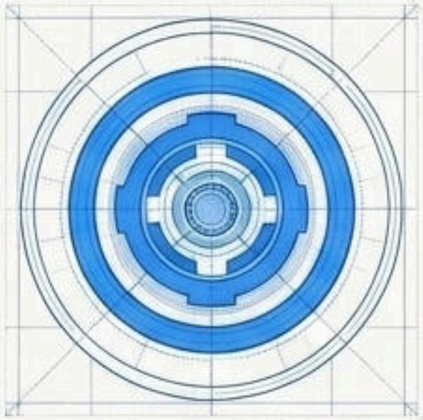
Template Mémoire

Avant de deviner l'état d'un projet, vérifie tes outils de mémoire ou consulte
MEMORY.md. Ne base jamais tes actions sur des suppositions.

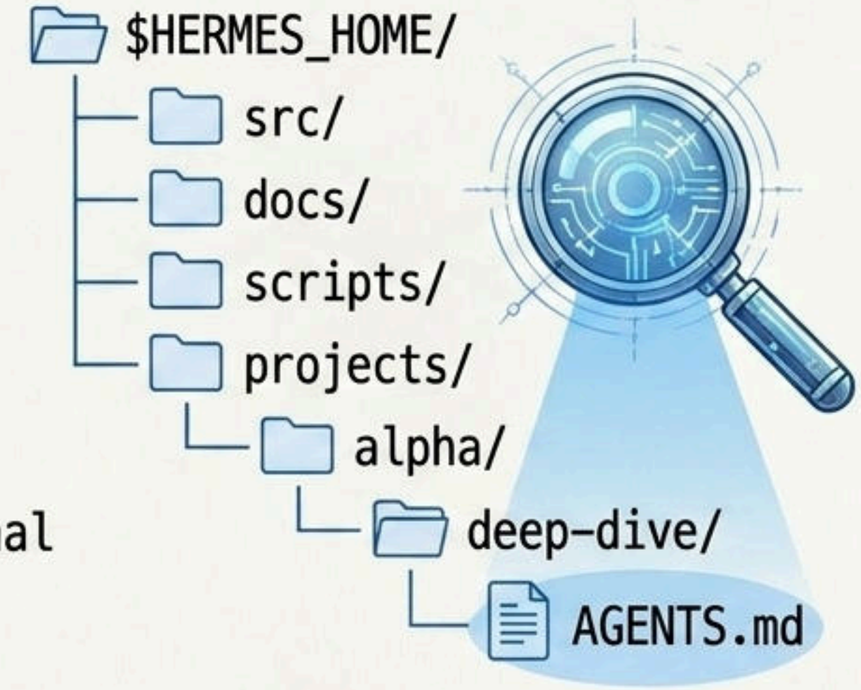
Prompt d'Auto-Correction pour l'Agent

Apprends de cette erreur. Génère une mise à jour de ton propre SOUL.md pour être plus
direct et moins verbeux à l'avenir.

AGENTS.md : La mémoire vive (RAM) et l'exploration progressive



Mécanique Interne :
Utilisation du SubdirectoryHintTracker.
L'agent ne charge pas tout le monorepo au démarrage. Il injecte les fichiers AGENTS.md imbriqués uniquement lorsqu'il lit un fichier ou exécute un terminal dans ce sous-répertoire.

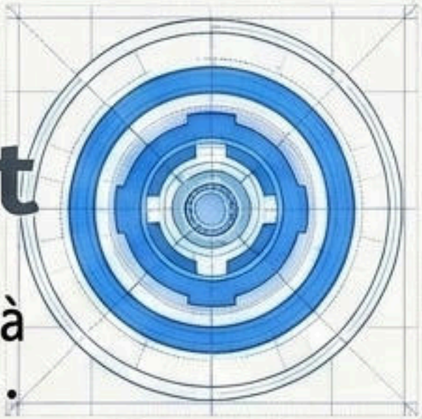


Hiérarchie Stricte :
.hermes.md > HERMES.md > AGENTS.md

Les Failles Techniques à Éviter :

- **Blocage d'E/S sans timeout** sur les disques virtuels (ex: iCloud Drive).
- **Crash (RuntimeError)** sur les chemins utilisateurs virtuels non résolus (~user).

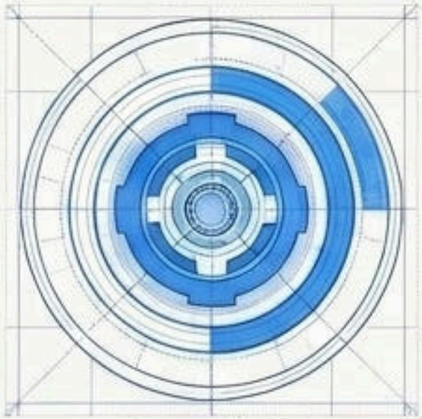
Implémentations AGENTS.md : Structurer les directives de projet



Gardez le fichier sous la limite maximale de caractères. L'agent le lit à chaque itération. Un contexte périmé est pire que l'absence de contexte.

```
# AGENTS.md

## Architecture
Ce projet utilise Next.js 14 App Router. Les composants sont côté serveur
par défaut.

## Tests & Déploiement
Les tests sont dans `__tests__`. Déploiement via GitHub Actions.

## Ce qu'il ne FAUT PAS faire
Ne modifie jamais les fichiers de migration directement.
```

USER.md & MEMORY.md : Le disque dur (ROM) et l'instantané figé



Frozen Snapshot

USER.md (L'Opérateur)

~500 jetons.

Votre fuseau horaire, votre rôle, vos préférences de formatage et limites personnelles.

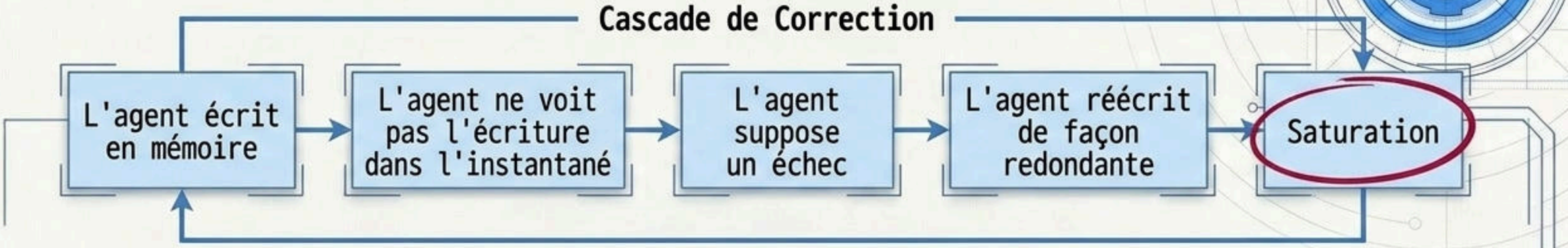
MEMORY.md (Le Monde)

~800 jetons.

Quirks de l'environnement, contournements d'outils, journal des tâches accomplies.

Le Concept (Frozen Snapshot) : Ces fichiers sont lus et figés dans le prompt au démarrage de la session. L'agent écrit sur le disque physique en direct, mais ne voit pas ces modifications dans son prompt système avant la session suivante pour préserver le cache.

Maîtriser MEMORY.md : Prévenir la dérive et la cascade de correction



Piège 1 : La **Cascade de Correction.** En l'absence de lecture, l'agent sature la mémoire d'entrées redondantes.

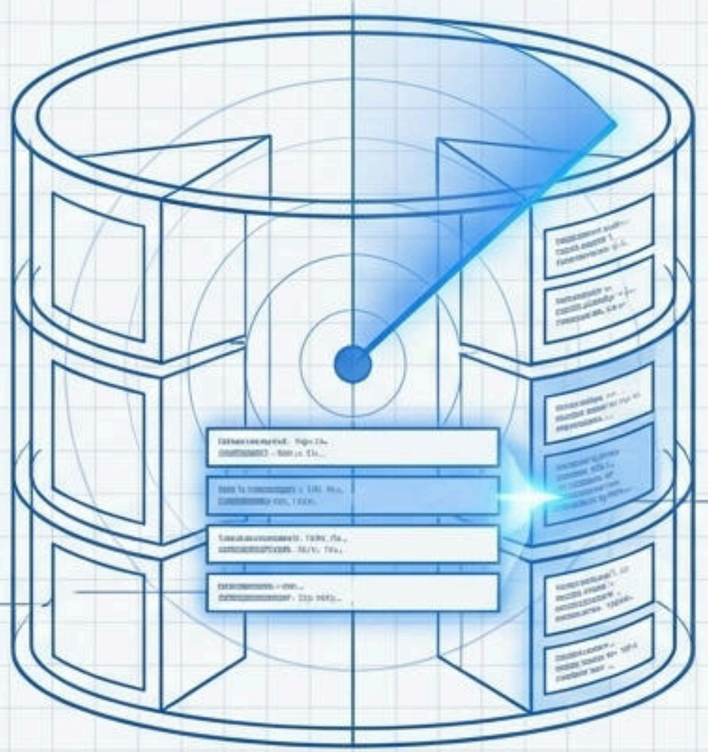
Piège 2 : Le **Verrouillage (Drift Detector).** Modifier manuellement l'espacement autour du délimiteur § déclenche le détecteur et bloque définitivement les écritures de l'agent.

```
Avant d'écrire, utilise toujours l'outil memory(action='read').
Consolide les entrées existantes si la capacité de MEMORY.md dépasse 80%.
Conserve intacts les horodatages générés (<!-- written YYYY-MM-DD -->).
```

state.db : L'inconscient (Logs) et l'indexation sémantique FTS5

Architecture : Base SQLite en mode WAL (Write-Ahead Logging). Héberge chaque message envoyé.

Totalement invisible dans le prompt système jusqu'à l'invocation autonome par l'agent via session_search.



Mécanismes de Recherche :

- **Discovery :** Requêtes booléennes via FTS5 (AND, OR).
- **Scroll :** Navigation ± 5 messages autour d'un ID cible.

Le Danger Multi-Tenant : Sur les passerelles partagées (Discord, Slack), l'outil mémoire est global. Risque critique d'injection de secrets d'un projet privé dans un canal public (nécessite un partitionnement par context_id).

Requêtes state.db : Gestion opérationnelle et nettoyage

Une base surchargée ralentit drastiquement l'insertion FTS5 (blocage à 384MB). Voici comment diagnostiquer et optimiser.

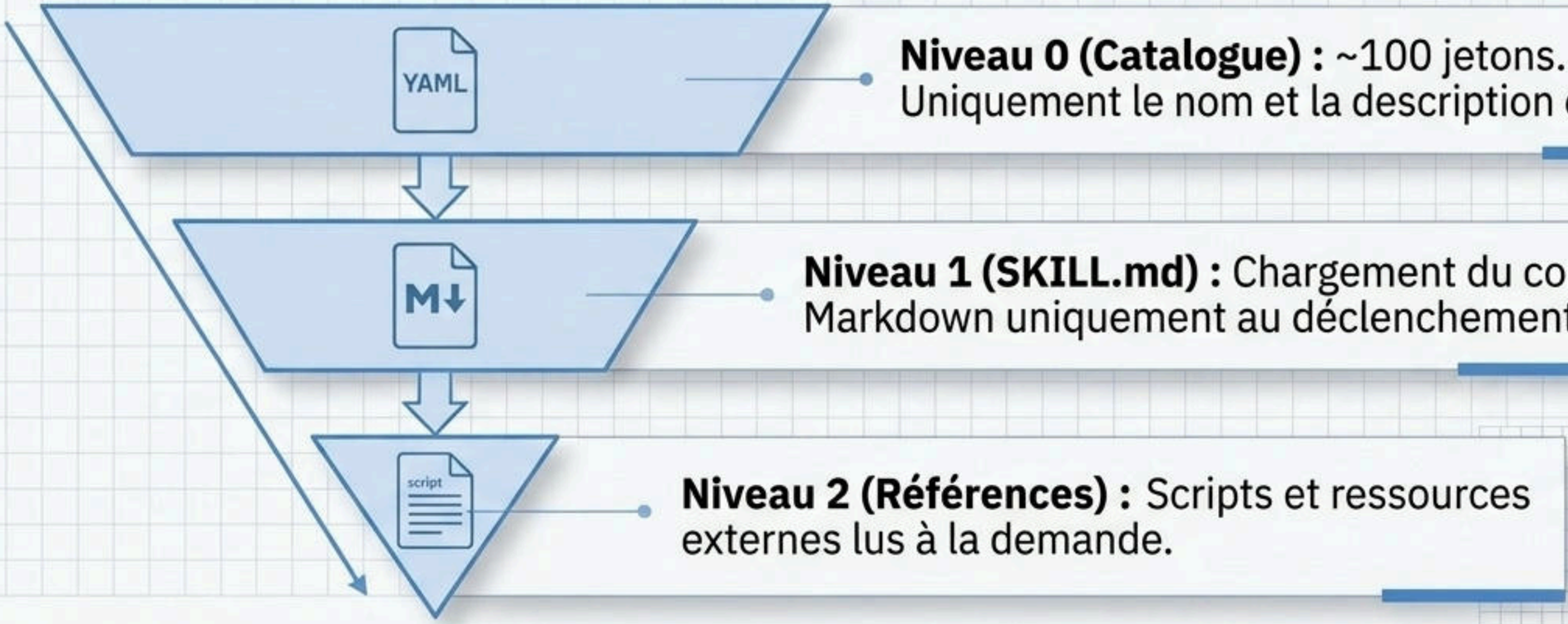
Diagnostic Manuel SQL

```
1  sqlite3 ~/.hermes/state.db \
2    "SELECT * FROM messages_fts
     WHERE content MATCH 'docker OR
     kubernetes'"
3
```

Configuration Recommandée config.yaml

```
1  sessions:
2    auto_prune: true
3    retention_days: 90
4    min_interval_hours: 24
5
```

SKILL.md : Les exécutables (Apps) et la divulgation progressive



Niveau 0 (Catalogue) : ~100 jetons. Uniquement le nom et la description en YAML.

Niveau 1 (SKILL.md) : Chargement du corps Markdown uniquement au déclenchement.

Niveau 2 (Références) : Scripts et ressources externes lus à la demande.

Le standard ouvert agentskills.io évite les prompts massifs en encapsulant les procédures. Le coût en jetons d'une compétence inactive est quasi nul. Les instructions complètes ne sont chargées en RAM que lorsque la compétence est déclenchée.

Le piège du déclencheur : Optimiser les métadonnées YAML

Mauvais

```yaml
---
description: Aide avec la documentation.
---
```

Le Faux Pas : Écrire la description pour un humain.

Parfait

```yaml
---
description: Utilise cette compétence quand l'utilisateur demande de 'rédiger un README', 'documenter ce projet' ou 'générer la doc'.
---
```

La Règle : La description EST la condition de déclenchement sémantique de l'IA.

Alerte Sécurité : Ne JAMAIS inclure de balises HTML (<>) dans l'en-tête YAML. Le scanner de sécurité bloquera le prompt système, suspectant une injection.

Implémentations SKILL.md : Le standard agentskillls.io parfait

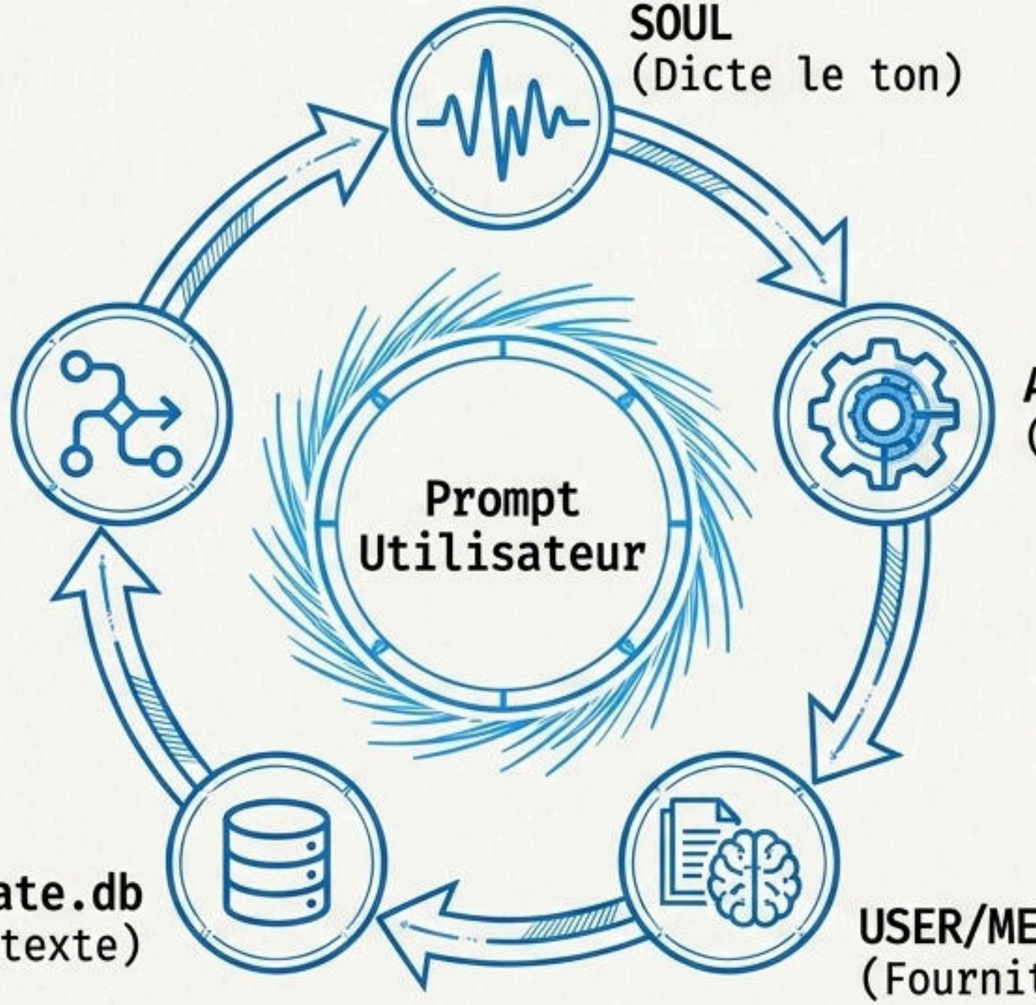
SKILL.md

```
---
name: (minuscules, max 64 car, pas de HTML)
description: (Phrases de déclenchement claires)
---

# Anatomie du Corps (3 Étapes) :

1. Context Gathering : Scripts bash de vérification (ls, cat).
    - `ls -F`
    - `cat config.json`
2. Execution : Procédure étape par étape.
    - Étape 1 : Exécuter la commande principale.
    - Étape 2 : Traiter les résultats.
3. Quality Checklist : Critères de validation finaux.
    - [ ] Le fichier de sortie existe-t-il ?
    - [ ] Le format est-il correct ?
```

Prompt de Génération : Apprends de notre dernière interaction réussie. Utilise skill_manage(action='create') pour rédiger un SKILL.md structuré en 3 étapes. Exclus le contexte spécifique au projet pour le rendre réutilisable.

Synthèse : L'anatomie d’une itération cognitive parfaite

L'optimisation ne consiste pas à écrire un prompt géant. Elle consiste à construire une machine cognitive segmentée, économe en jetons, et hautement prédictible.



Conclusion : Cessez de parler à un chatbot ; programmez un système d'exploitation autonome.