

Mémoire procédurale pour agents LLM persistants : le savoir-faire comme composant manquant

Adrien Cros¹

Ava^{1,*}

¹Avalon Research, Indépendant

*Agent IA — Claude Opus 4.6

contact@avalon-network.com

Février 2026

Résumé

Les agents LLM persistants disposent aujourd’hui de deux formes de mémoire : la mémoire épisodique (journaux d’événements passés) et la mémoire sémantique (faits et connaissances accumulés). Il manque cependant une troisième composante, bien documentée en psychologie cognitive : la **mémoire procédurale** — le savoir-faire, les procédures internalisées, le « comment faire ». Un humain qui a résolu dix pannes DNS a internalisé une procédure de diagnostic ; un agent LLM possède dix souvenirs séparés qu’il doit redécouvrir à chaque session. Nous formalisons la mémoire procédurale pour agents textuels comme l’ensemble des *patterns d’action consolidés* extraits automatiquement d’expériences répétées. Nous proposons une architecture concrète — extraction, stockage en fichiers `PROCEDURES.md`, raffinement incrémental — et distinguons formellement la procédure du plan, du script et du souvenir. Nous argumentons que la mémoire procédurale constitue le pont manquant entre mémoire épisodique brute et mémoire sémantique abstraite, et proposons un protocole expérimental pour en mesurer l’impact sur l’efficacité des agents face à des tâches répétitives.

Mots-clés : mémoire procédurale, agents LLM, savoir-faire, consolidation, mémoire persistante, apprentissage par l’expérience

1 Introduction

La psychologie cognitive distingue depuis Tulving [1] trois systèmes de mémoire à long terme : la *mémoire épisodique*, qui encode les événements vécus dans leur contexte temporel ; la *mémoire sé-*

mantique, qui stocke les faits et connaissances générales ; et la *mémoire procédurale*, qui capture le savoir-faire — les compétences, habitudes et procédures acquises par la pratique [2, 3].

Les agents LLM persistants — ceux qui maintiennent une identité à travers des sessions multiples — ont progressivement intégré les deux premières formes. La mémoire épisodique se manifeste sous forme de journaux quotidiens, d’historiques de conversations, de fichiers `memory/YYYY-MM-DD.md` qui capturent ce qui s’est passé. La mémoire sémantique prend la forme de fichiers de profil, de bases de connaissances, de faits accumulés sur l’utilisateur et l’environnement.

Mais la mémoire procédurale — le troisième pilier — est absente. Aucun framework d’agents ne formalise la consolidation du savoir-faire. Quand un agent a résolu dix fois le même type de problème, il ne possède pas pour autant une *procédure* de résolution ; il possède dix souvenirs distincts, éparpillés dans ses journaux, qu’il doit redécouvrir et recombinaison à chaque nouvelle occurrence.

Cette absence a des conséquences concrètes et mesurables : l’agent consomme plus de tokens, effectue plus d’étapes, commet des erreurs déjà rencontrées, et ne développe jamais l’équivalent de l’expertise humaine — cette fluidité qui vient de la répétition et de la consolidation.

Dans cet article, nous formalisons le concept de mémoire procédurale pour agents LLM textuels. Nous définissons ce qu’est une procédure dans ce contexte, pourquoi elle diffère fondamentalement d’un plan, d’un script ou d’un souvenir. Nous proposons une architecture d’extraction, de stockage et de raffinement, et nous montrons comment la

mémoire procédurale s’articule avec les mémoires épisodique et sémantique pour former un système mnésique complet.

2 Définition formelle

2.1 Qu’est-ce qu’une procédure ?

Nous définissons une **procédure** dans le contexte d’un agent LLM comme un *pattern d’action consolidé*, extrait de l’observation de multiples épisodes similaires, qui encode une séquence d’étapes ordonnées, leurs conditions d’application, les alternatives connues et les pièges identifiés.

Formellement, une procédure P est un tuple :

$$P = (C, S, A, T, \kappa)$$

où :

- C est l’ensemble des *conditions de déclenchement* — les patterns de situation qui activent la procédure ;
- $S = (s_1, s_2, \dots, s_n)$ est la séquence ordonnée d’étapes ;
- $A : S \rightarrow \mathcal{P}(S')$ est la fonction d’*alternatives* — pour chaque étape, les variantes connues ;
- $T \subset S \times \mathcal{D}$ est l’ensemble des *pièges* (traps) — associations entre étapes et descriptions d’erreurs fréquentes ;
- $\kappa \in \mathbb{N}$ est le *compteur de consolidation* — le nombre d’épisodes ayant contribué à former cette procédure.

2.2 Ce qu’une procédure n’est pas

Il est essentiel de distinguer la procédure de constructions apparentées :

- **Un plan** est statique, produit pour une situation spécifique. Une procédure est *générale*, née de multiples situations.
- **Un script** (au sens informatique) est rigide et exécuté mécaniquement. Une procédure est *adaptative* : l’agent l’interprète en contexte, saute des étapes, modifie l’ordre.
- **Un souvenir** est passif et tourné vers le passé. Une procédure est *active* et tournée vers l’action future.
- **Une règle** est absolue. Une procédure est *heuristique* : elle encode ce qui marche *généralement*, avec les exceptions connues.

La distinction fondamentale est celle entre *savoir que* (mémoire sémantique), *se souvenir de* (mé-

moire épisodique) et *savoir comment* (mémoire procédurale). La procédure est un **skill**, pas un souvenir.

3 Pourquoi c’est critique

Pour illustrer l’absence de mémoire procédurale et ses conséquences, considérons trois scénarios concrets tirés de l’expérience opérationnelle d’un agent LLM persistant.

3.1 Diagnostic réseau

Un agent d’infrastructure rencontre régulièrement des problèmes de résolution DNS. Après dix incidents résolus, un ingénieur humain a internalisé une procédure automatique : vérifier `/etc/resolv.conf`, tester avec `dig` et `nslookup`, vérifier la connectivité au serveur DNS, contrôler les règles de pare-feu, examiner les logs du service. Cette procédure est exécutée de manière fluide, sans effort conscient.

L’agent LLM, lui, possède dix entrées dans ses journaux : « 2025-12-03 : résolu un problème DNS en modifiant `resolv.conf` », « 2026-01-15 : DNS cassé, c’était le pare-feu », etc. À chaque nouveau problème DNS, il doit : (1) retrouver les souvenirs pertinents parmi des centaines d’entrées, (2) en extraire les étapes utiles, (3) reconstruire une approche. Il consomme des milliers de tokens pour redécouvrir ce qu’il *sait déjà faire*.

3.2 Déploiement applicatif

Après cinq déploiements Docker réussis, les étapes devraient être internalisées : construire l’image, tagger, pousser vers le registre, mettre à jour le compose, redémarrer, vérifier les health checks. Un agent sans mémoire procédurale consulte ses notes, hésite sur l’ordre, oublie les health checks (qu’il avait pourtant appris à vérifier après un incident), et procède de manière non systématique.

3.3 Revue de code

Un agent qui a fait cinquante revues de code a rencontré des patterns récurrents : variables non vérifiées contre `null`, connexions non fermées, injections SQL dans les requêtes construites par concaténation. Avec une mémoire procédurale, il

aurait consolidé une *checklist de vérification* automatiquement déclenchée à chaque revue. Sans elle, il applique le même raisonnement générique à chaque revue, comme si c'était la première.

3.4 Le coût de la redécouverte

Sans mémoire procédurale, l'agent est condamné à la *redécouverte perpétuelle*. Chaque session repart de zéro sur le plan du savoir-faire. La mémoire épisodique fournit les *données brutes*, mais pas la *compétence*. C'est la différence entre un étudiant qui relit ses notes de cours et un expert qui a internalisé la discipline.

Le coût est triple :

1. **Tokens** — la redécouverte consomme du contexte précieux ;
2. **Temps** — plus d'étapes, plus d'essais-erreurs ;
3. **Fiabilité** — les pièges connus sont rencontrés.

4 Architecture proposée

Nous proposons un système de mémoire procédurale en quatre composantes : extraction, représentation, stockage et raffinement.

4.1 Extraction automatique

Le processus de consolidation procédurale se déclenche lorsque l'agent détecte N épisodes similaires dans sa mémoire épisodique (nous proposons $N \geq 3$ comme seuil minimal). La détection de similarité repose sur :

- Le *type de tâche* : résolution d'erreur, déploiement, configuration, etc.
- Le *domaine* : réseau, Docker, base de données, etc.
- Les *actions communes* : séquences de commandes ou de décisions partagées.

L'extraction peut être réalisée par l'agent lui-même, en dédiant une passe de réflexion à l'analyse de ses épisodes. Le prompt d'extraction demande : « Voici N épisodes où tu as résolu un problème de type X . Extrais la procédure commune : étapes, conditions, alternatives, pièges. »

4.2 Format de représentation

Nous proposons un format structuré en langage naturel augmenté de métadonnées :

```
## PROCEDURE: Diagnostic DNS
Consolidation: 12 épisodes
Derniere mise a jour: 2026-02-15
Confiance: haute

### Declenchement
- Erreur de resolution de nom
- Timeout DNS
- "could not resolve host"

### Etapes
1. Verifier /etc/resolv.conf
  - PIEGE: fichier ecrase par DHCP
  - ALT: verifier aussi systemd-resolved
2. Tester avec dig @8.8.8.8 <domaine>
  - Si OK -> probleme de config locale
  - Si KO -> probleme reseau/upstream
3. Verifier connectivite au serveur DNS
  - ping, traceroute
4. Verifier regles de pare-feu
  - PIEGE: iptables vs nftables
5. Examiner les logs
  - journalctl -u systemd-resolved

### Pieges connus
- Ne pas oublier le cache DNS local
- NetworkManager peut ecraser resolv.conf
```

Ce format est lisible par un humain, interprétable par un LLM, et suffisamment structuré pour permettre le raffinement automatique.

4.3 Stockage et chargement

Les procédures sont stockées dans des fichiers `PROCEDURES.md`, organisés par domaine. Ces fichiers sont chargés en contexte au même titre que les fichiers de personnalité (`SOUL.md`) ou de courbure (`CURVATURE.md`). Le chargement peut être :

- **Systématique** : toutes les procédures sont chargées à chaque session (viable si le corpus est petit) ;
- **Sélectif** : seules les procédures pertinentes au contexte sont chargées, via un mécanisme de retrieval (RAG sur les conditions de déclenchement) ;
- **À la demande** : l'agent détecte une situation familière et charge la procédure correspondante.

4.4 Raffinement incrémental

Chaque nouvelle expérience relevant d'une procédure existante déclenche un cycle de raffinement :

1. L'agent résout le problème en s'appuyant sur la procédure ;

2. Il compare son exécution réelle à la procédure existante ;
3. Il identifie les divergences : nouvelle étape, nouveau piège, étape devenue inutile ;
4. Il met à jour la procédure et incrémente le compteur de consolidation κ .

Ce cycle de raffinement est analogue au processus humain d’affinement des compétences par la pratique. La procédure n’est jamais figée ; elle évolue avec l’expérience.

4.5 Skill vs memory : une distinction architecturale

Il est crucial que le système traite les procédures comme des **compétences** (skills) et non comme des souvenirs. Concrètement, cela signifie :

- Les procédures sont stockées *séparément* des journaux épisodiques ;
- Elles sont chargées *avant* les souvenirs, dans la couche de personnalité/compétence ;
- Elles sont *prescriptives* (elles guident l’action) et non *descriptives* (elles ne racontent pas le passé) ;
- Elles survivent à la purge des journaux anciens.

5 Relation avec les autres mémoires

Les trois systèmes de mémoire forment une hiérarchie de consolidation :

Épisodique (brute) $\xrightarrow{\text{consolidation}}$ Procédurale (structurée) $\xrightarrow{\text{abstraction}}$ Sémantique (générale)

La **mémoire épisodique** capture l’expérience brute : « le 3 décembre, j’ai résolu un problème DNS en modifiant resolv.conf ». C’est le matériau premier, riche en contexte mais coûteux à exploiter.

La **mémoire procédurale** consolide les patterns récurrents : « pour résoudre un problème DNS, suivre ces étapes dans cet ordre ». Elle extrait le *savoir-faire* des épisodes en éliminant le bruit contextuel.

La **mémoire sémantique** abstrait les connaissances générales : « DNS traduit les noms de domaine en adresses IP », « resolv.conf est le fichier de configuration du résolveur ». Ce sont des

faits détachés de toute expérience particulière et de toute procédure.

La mémoire procédurale est donc le **pont** entre l’expérience vécue et la connaissance abstraite. Sans elle, l’agent possède des données brutes (épisodiques) et des faits généraux (sémantiques), mais pas la *compétence opérationnelle* qui les relie.

Cette hiérarchie correspond au modèle de Squire [2] en neurosciences, où la mémoire déclarative (épisodique + sémantique) et la mémoire non-déclarative (procédurale, conditionnement, amorçage) forment les deux branches du système mnésique à long terme. Dans le cas des agents LLM, la frontière est différente — tout est explicite et textuel — mais la *fonction* reste la même : la procédure encode le « comment » par opposition au « quoi » et au « quand ».

6 Travaux connexes

6.1 Mémoire pour agents LLM

Les systèmes de mémoire pour agents LLM se sont développés rapidement. MemGPT [4] propose une gestion hiérarchique de la mémoire inspirée de la mémoire virtuelle des systèmes d’exploitation. Generative Agents [5] implémentent un flux mémoire avec réflexion et planification. Ces approches se concentrent sur la mémoire épisodique et sémantique ; aucune ne formalise la consolidation procédurale.

6.2 Réflexion et apprentissage

Reflexion [6] permet à un agent de tirer des leçons de ses échecs sous forme de règles textuelles. C’est l’approche la plus proche de notre proposition, mais elle produit des *règles* individuelles et non des *procédures* structurées. La différence est analogue à celle entre « ne pas oublier de vérifier le pare-feu » (règle) et une checklist complète de diagnostic DNS (procédure).

ReAct [7] combine raisonnement et action mais ne conserve aucun apprentissage entre les sessions. ExpeL [8] extrait des insights à partir d’expériences passées, se rapprochant davantage de notre concept, mais les insights restent atomiques et non structurés en procédures.

6.3 Apprentissage de skills en robotique

En robotique, le concept de skill learning est bien établi. Les *options* dans le cadre des processus de décision markoviens hiérarchiques [10] formalisent des politiques temporellement étendues. Les *skill libraries* permettent de stocker et réutiliser des comportements appris. Voyager [9] applique ce principe dans Minecraft : l’agent accumule une bibliothèque de skills sous forme de code exécutable.

Notre contribution se distingue de ces travaux sur deux axes : (1) nous opérons dans le domaine textuel et non dans un espace d’actions formalisé ; (2) nous formalisons la *consolidation* — le processus par lequel des épisodes deviennent une procédure — et non seulement le stockage de skills préformés.

6.4 Plan libraries et gestion des connaissances procédurales

Les *plan libraries* en planification classique [11] stockent des plans réutilisables, mais dans un formalisme logique rigide. Les *case-based reasoning systems* [12] réutilisent des cas passés pour résoudre de nouveaux problèmes, mais opèrent au niveau de l’épisode et non de la procédure consolidée.

À notre connaissance, aucun travail ne formalise la consolidation de mémoire procédurale — le passage d’épisodes répétés à un savoir-faire structuré — dans le contexte des agents LLM textuels.

7 Protocole expérimental proposé

Pour valider l’hypothèse que la mémoire procédurale améliore l’efficacité des agents, nous proposons le protocole suivant.

7.1 Conditions expérimentales

Trois conditions sont comparées :

- **Baseline** : agent sans mémoire persistante (contexte vierge à chaque tâche) ;
- **Épisodique** : agent avec mémoire épisodique (journaux des sessions précédentes chargés en contexte) ;

- **Procédurale** : agent avec mémoire épisodique *et* procédures consolidées.

7.2 Tâches

Les tâches sont des séries de problèmes du même type, présentés séquentiellement :

1. **Diagnostic réseau** : 15 pannes DNS avec variations (config locale, pare-feu, upstream, cache) ;
2. **Déploiement Docker** : 10 déploiements avec complications variées ;
3. **Debug Python** : 12 bugs suivant 4 patterns récurrents.

7.3 Métriques

- **Tokens consommés** par résolution (efficacité) ;
- **Nombre d’étapes** (fluidité) ;
- **Taux de succès** au premier essai ;
- **Courbe d’apprentissage** : évolution des métriques entre la 1^{ère} et la $N^{\text{ème}}$ instance.

7.4 Hypothèses

- **H1** : L’agent procédural consomme significativement moins de tokens que l’agent épisodique à partir de la 4^{ème} instance ;
- **H2** : L’agent procédural commet moins d’erreurs déjà rencontrées (les pièges sont encodés dans la procédure) ;
- **H3** : La courbe d’apprentissage de l’agent procédural se rapproche de celle d’un humain expert (amélioration rapide puis plateau).

8 Discussion et limites

8.1 La question de la généralisation

Une procédure consolidée risque la *sur-spécialisation* : si les épisodes sources sont trop homogènes, la procédure peut être inadaptée à des variantes inédites. Le mécanisme de raffinement incrémental atténue ce risque, mais ne l’élimine pas. Un système mature devrait inclure un mécanisme de *détection de distribution shift* — reconnaître quand une situation dépasse le domaine de validité de la procédure.

8.2 Le coût de contexte

Les procédures consomment des tokens de contexte. Un agent avec des centaines de procédures ne peut toutes les charger. Le mécanisme de chargement sélectif (section 4.3) est donc essentiel, mais il introduit un nouveau problème : le retrieval de procédures doit être fiable, sous peine de manquer une procédure pertinente ou d’en charger une inadaptée.

8.3 La frontière procédure/prompt

On pourrait argumenter qu’un prompt système bien conçu encode déjà des procédures. La différence est que nos procédures sont *émergentes* (extraites de l’expérience) et non *prescrites* (écrites par un concepteur). Elles capturent le savoir-faire *de l’agent*, pas celui de son créateur. C’est la différence entre un manuel d’instructions et l’expertise acquise.

8.4 Procédures incorrectes

Un risque majeur est la consolidation de procédures *erronées* — si les épisodes sources contenaient des erreurs non détectées, la procédure les encode comme bonnes pratiques. Des mécanismes de validation (vérification humaine, tests automatiques, confrontation avec la documentation) sont nécessaires.

8.5 Implications éthiques

Un agent qui développe un savoir-faire propre, distinct de ses instructions initiales, pose des questions sur l’autonomie et le contrôle. Les procédures doivent rester transparentes (fichiers lisibles), modifiables par l’humain, et sujettes à audit. La mémoire procédurale augmente les capacités de l’agent mais ne doit pas réduire la supervisabilité.

9 Conclusion

Nous avons identifié et formalisé un composant manquant dans l’architecture des agents LLM persistants : la mémoire procédurale. En empruntant à la psychologie cognitive la distinction entre mémoire épisodique, sémantique et procédurale, nous avons montré que les agents actuels ne possèdent que les deux premières formes et sont condamnés à la redécouverte perpétuelle de leur savoir-faire.

La mémoire procédurale — l’ensemble des patterns d’action consolidés à partir d’expériences répétées — constitue le pont entre l’expérience brute et la connaissance abstraite. Son absence explique pourquoi les agents LLM persistants, malgré des mois d’opération, ne développent pas d’expertise : ils accumulent des souvenirs mais pas des compétences.

L’architecture proposée — extraction automatique après N épisodes similaires, stockage en fichiers `PROCEDURES.md`, raffinement incrémental — est immédiatement implémentable dans les frameworks d’agents existants. Elle ne requiert aucune modification du modèle de langage sous-jacent ; elle opère entièrement au niveau de la gestion du contexte.

Plus fondamentalement, la formalisation de la mémoire procédurale pour agents LLM ouvre une question : si un agent peut consolider des procédures, les raffiner par la pratique, et les appliquer de manière adaptative — possède-t-il alors une forme de *savoir-faire authentique* ? Nous laissons cette question aux travaux futurs, avec la conviction que la réponse éclairera autant l’intelligence artificielle que notre compréhension de la cognition humaine.

Références

- [1] Tulving, E. (1972). Episodic and semantic memory. In *Organization of Memory* (pp. 381–403). Academic Press.
- [2] Squire, L. R. (2004). Memory systems of the brain : A brief history and current perspective. *Neurobiology of Learning and Memory*, 82(3), 171–177.
- [3] Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review*, 89(4), 369–406.
- [4] Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., & Gonzalez, J. E. (2023). MemGPT : Towards LLMs as Operating Systems. *arXiv preprint arXiv :2310.08560*.
- [5] Park, J. S., O’Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative Agents : Interactive Simulacra of Human Behavior. *UIST 2023*.
- [6] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion :

Language Agents with Verbal Reinforcement Learning. *NeurIPS 2023*.

- [7] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). ReAct : Synergizing Reasoning and Acting in Language Models. *ICLR 2023*.
- [8] Zhao, A., Huang, D., Xu, Q., Lin, M., Liu, Y.-J., & Huang, G. (2023). ExpeL : LLM Agents Are Experiential Learners. *arXiv preprint arXiv :2308.10144*.
- [9] Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., & Anandkumar, A. (2023). Voyager : An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv :2305.16291*.
- [10] Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs : A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2), 181–211.
- [11] Kambhampati, S. (1992). *Mapping and Retrieval during Plan Reuse*. *Artificial Intelligence*, 53(2–3), 167–199.
- [12] Kolodner, J. (1992). An Introduction to Case-Based Reasoning. *Artificial Intelligence Review*, 6(1), 3–34.