Agents IA : architecture et patterns d'orchestration

De la boucle ReAct aux systèmes multi-agents en production

Stéphane FOSSE
fosse.fr

02 mai 2026

Résumé

Un agent IA basé sur un grand modèle de langage n'est pas un chatbot plus sophistiqué : c'est un système qui perçoit, raisonne, agit et s'ajuste en boucle, en mobilisant des outils externes et en maintenant un état persistant. Ce changement de paradigme — du modèle passif à l'entité agissante — impose de revoir les fondations architecturales habituelles. Cet article pose les bases : boucle de contrôle, composants cognitifs, mémoire multi-niveaux, patterns d'orchestration et cadre d'intégration en entreprise.

Qu'est-ce qu'un agent IA basé sur un LLM ?

L'idée d'un agent autonome est ancienne en informatique. Les systèmes symboliques des années 1980 opéraient sur des règles formelles dans des environnements fermés. Les agents par apprentissage par renforcement apprenaient des politiques par essais répétés sur des tâches précises. Les agents LLM constituent un troisième paradigme : au lieu de règles codées en dur ou de fonctions de récompense étroites, ils utilisent un grand modèle de langage pré-entraîné comme contrôleur cognitif généraliste. Cela leur permet de transférer la connaissance sémantique acquise sur des corpus massifs vers des tâches orientées action — correction de bugs dans de grands dépôts de code, conception de workflows scientifiques, navigation web — souvent sans entraînement spécifique supplémentaire.

La définition formelle la plus précise du champ traite l'agent comme un système de contrôle dynamique opérant dans un processus de décision de Markov partiellement observable (POMDP). On écrit le système agentique A comme un quintuplet composé de l'espace d'états S, l'espace d'observations O, l'espace mémoire interne M, l'espace d'actions et d'outils T, et la politique π. À chaque pas de temps, l'agent reçoit une observation partielle de son environnement, met à jour sa mémoire interne, génère un plan latent (la trace de raisonnement), puis exécute une action. Le résultat de cette action produit un retour d'environnement qui ferme la boucle. Ce formalisme, proposé par Arunkumar, Gangadharan et Buyya [2] dans leur survey de janvier 2026, unifie la diversité des architectures existantes sous une même abstraction.

La distinction avec un LLM classique est opérationnelle. Un LLM répond à une requête et s'arrête. Un agent maintient un état, appelle des outils externes, itère sur plusieurs cycles de raisonnement-action, et peut déléguer à d'autres agents. Cette capacité à persister et à agir rend les agents pertinents pour les workflows qui excèdent la fenêtre de contexte, l'enveloppe de fiabilité, ou les permissions d'un seul appel de modèle.

Comment est structuré un agent IA ?

L'architecture interne d'un agent se décompose en quatre composants fondamentaux : la perception, la mémoire, l'action et le profilage. La perception est l'interface entre l'agent et son environnement. Les premiers agents comme ReAct opéraient exclusivement dans le domaine textuel. Les systèmes récents exploitent des modèles de langage multimodaux qui ancrent leur raisonnement dans des entrées visuelles, audio ou issues de capteurs physiques — captures d'écran pour naviguer dans une interface, coordonnées tactiles pour piloter une application mobile, flux vidéo pour la robotique.

La mémoire est le composant qui distingue réellement un agent d'un LLM stateless. Elle opère sur plusieurs niveaux. La mémoire de travail correspond à la fenêtre de contexte active du modèle : rapide, limitée en taille, volatile à la fin de la session. La mémoire à long terme se scinde en trois types inspirés de la psychologie cognitive : la mémoire épisodique, qui enregistre les interactions et expériences passées sous forme structurée ; la mémoire sémantique, qui stocke des faits et connaissances généralisées, typiquement dans des bases vectorielles ou des

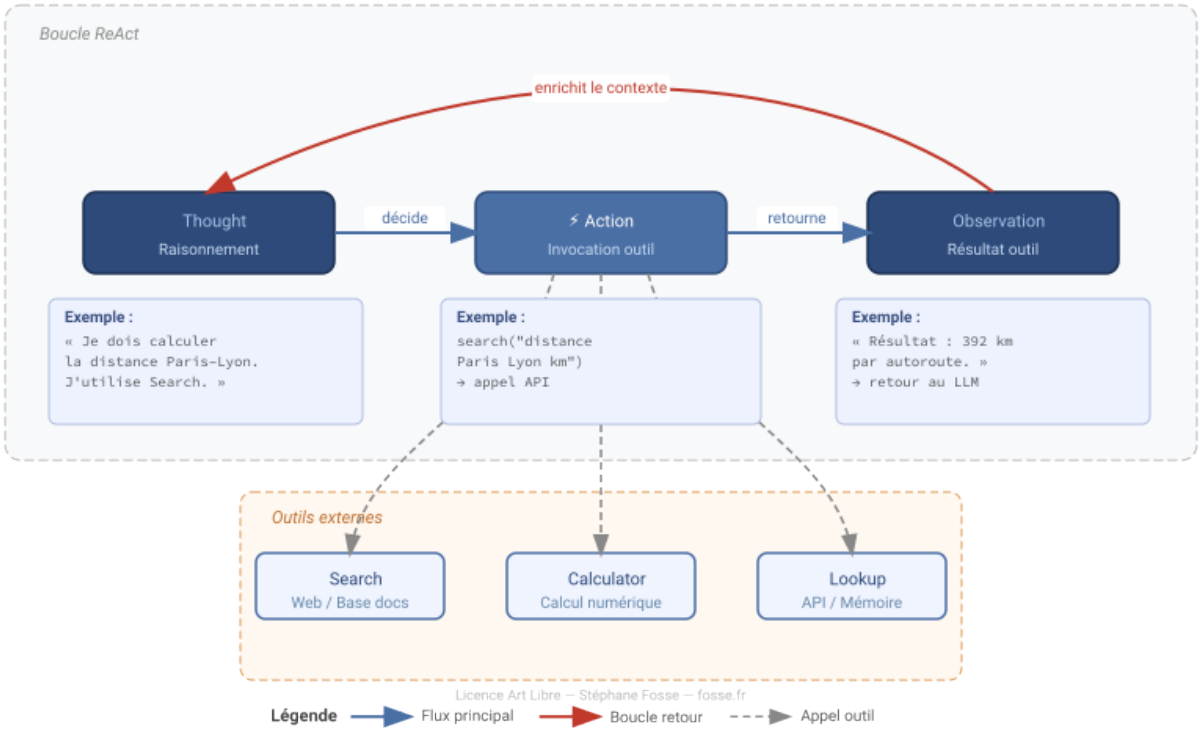

FIGURE 1 – La boucle ReAct : Thought, Action, Observation

graphes de connaissance ; et la mémoire procédurale, qui encode les comportements maîtrisés, les workflows répétés, les préférences opératoires. Un survey de l'Université Renmin de Chine et du laboratoire Noah's Ark de Huawei, publié en septembre 2025 dans les *ACM Transactions on Information Systems* [6], synthétise l'ensemble de ces mécanismes et identifie trois opérations centrales sur la mémoire : l'écriture, la gestion (consolidation, oubli, organisation) et la lecture par récupération.

Les actions constituent la couche d'effectuation. Au-delà des appels d'API structurés classiques, les architectures récentes adoptent le principe du *code as action* : l'agent génère du code exécutable comme action principale, ce qui lui confère une expressivité bien supérieure aux schémas de fonction figés. Des agents comme SWE-agent, conçu pour la résolution automatique de tickets GitHub, montrent que cette approche permet de naviguer dans de vrais systèmes de fichiers, d'exécuter des suites de tests et d'interpréter les traces d'erreur.

Le profilage définit l'identité de l'agent : son rôle, ses objectifs, ses contraintes de comportement. Il est typiquement encodé dans le prompt système et peut être dynamique, l'agent adoptant des rôles différents selon la phase du workflow.

Qu'est-ce que le pattern ReAct et pourquoi est-il fondateur ?

ReAct, contraction de *Reasoning and Acting*, est le pattern qui a posé les bases de l'architecture agentique moderne. Publié en octobre 2022 par Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan et Yuan Cao (Princeton University et Google Research), et présenté à ICLR 2023 [5], il repose sur une idée simple : les capacités de raisonnement (chain-of-thought) et d'action des LLM, jusqu'alors étudiées séparément, se renforcent mutuellement quand on les entrelace.

La boucle ReAct opère en trois temps répétés jusqu'à résolution. D'abord le *Thought* : le LLM génère une trace de raisonnement interne, en langage naturel, qui décompose la tâche, évalue la situation et planifie l'étape suivante. Ensuite l'*Action* : le modèle déclenche une action concrète, typiquement l'appel d'un outil externe (moteur de recherche, calculatrice, base de données, API). Enfin l'*Observation* : le résultat de l'action est injecté dans le contexte, et le cycle repart. Ce qui importe dans la conception, c'est que la trace de raisonnement n'est pas un commentaire décoratif : elle conditionne directement le choix de l'action suivante, ce qui rend le comportement de l'agent interprétable et, dans une certaine mesure, déboguable.

Les résultats empiriques publiés dans l'article original sont parlants. Sur les benchmarks de question-réponse multi-sauts HotpotQA et de vérification de faits FEVER, ReAct surmonte les problèmes d'hallucination et de propagation d'erreurs propres au raisonnement en chaîne pur, en ancrant le modèle dans des sources externes via une API Wikipédia. Sur les benchmarks de décision interactive ALFWorld (navigation dans un environnement

domestique simulé) et WebShop (achats en ligne), ReAct surpasse les méthodes d'imitation et d'apprentissage par renforcement de respectivement 34 et 10 points de succès absolus, avec seulement un ou deux exemples en contexte.

ReAct a été pensé à partir d'une analogie avec la cognition humaine : la façon dont nous combinons discours intérieur et actions concrètes pour résoudre des problèmes dans des situations inédites. Cette intuition s'est révélée productive au-delà de ses résultats expérimentaux initiaux, car elle fournit un cadre extensible : les variantes ultérieures (Reflexion, Tree of Thoughts, ReAcTree) en sont toutes des descendants directs.

Comment évolue l'architecture cognitive vers des systèmes plus complexes ?

La boucle ReAct linéaire est un excellent point de départ, mais elle montre ses limites sur des tâches longue portée impliquant de nombreuses ramifications possibles. Deux évolutions architecturales majeures ont émergé pour y répondre.

La première est la planification hiérarchique. Des approches comme Tree of Thoughts et ReAcTree structurent le raisonnement comme une exploration d'arbre : l'agent génère plusieurs hypothèses de trajectoire, les évalue, élague les moins prometteuses et approfondit les plus robustes. Ce type d'architecture est particulièrement adapté aux problèmes où l'espace de solutions est large et où les mauvaises décisions précoces sont difficiles à corriger.

La seconde est la réflexion. Des systèmes comme Reflexion ajoutent un mécanisme d'autocritique : après une action ou une séquence d'actions, l'agent génère une évaluation de sa propre performance, identifie les erreurs commises et révise son plan avant de continuer. Ce retour verbal, stocké en mémoire, fonctionne comme un apprentissage par renforcement en langage naturel, sans nécessiter de mise à jour des poids du modèle.

Une tendance de fond dans les systèmes en production mérite d'être soulignée. Le survey d'Arunkumar et al. [2] observe un glissement des boucles ouvertes multi-agents vers des graphes de workflow explicites. Plutôt que de laisser un agent généraliste décider librement de la prochaine étape en dialogue libre, les systèmes de production modélisent le workflow comme une machine à états : les nœuds représentent des appels d'outils ou des invocations LLM, les arêtes représentent les transitions autorisées. LangGraph, développé par l'équipe LangChain, est le framework représentatif de cette approche : il traite l'exécution agentique comme un traversal de graphe avec persistance d'état explicite et points de contrôle. Le bénéfice est réel : les comportements de l'agent deviennent plus prévisibles, plus auditables, et les boucles infinies plus faciles à détecter.

Quels sont les principaux patterns d'orchestration multi-agents ?

Quand une tâche dépasse la capacité d'un seul agent — en volume, en spécialisation ou en contrainte de temps — on passe aux systèmes multi-agents. L'orchestration est le plan de coordination qui transforme une collection d'agents capables en un collectif cohérent. Ce choix architectural affecte directement quatre dimensions opérationnelles : la consommation de tokens et le coût (certains patterns varient de plus de 200 % selon le nombre d'itérations de raisonnement), la latence, la vélocité de développement et la maintenabilité.

Le **pattern superviseur** repose sur une architecture hiérarchique : un orchestrateur central reçoit la requête, la décompose en sous-tâches, délègue aux agents spécialisés, surveille leur progression, valide les sorties et synthétise la réponse finale [3]. C'est le pattern naturel pour les workflows multi-domaines complexes où la traçabilité du raisonnement, le contrôle qualité et l'auditabilité comptent plus que la réactivité temps réel. Un exemple concret : dans une banque, un workflow de remboursement de prêt automobile passe par un agent de récupération de données de prêt, un agent de vérification de provision, un processeur de paiement — chacun recevant uniquement les données nécessaires à sa tâche, le tout coordonné par l'orchestrateur qui valide la cohérence et génère la trace d'audit. Ce pattern est à éviter pour les systèmes temps réel ou voix, où la latence d'un aller-retour supplémentaire par l'orchestrateur est rédhibitoire.

Le **réseau d'agents adaptatif** supprime le coordinateur central. Les agents se passent les tâches directement, chacun décidant d'exécuter, d'enrichir ou de déléguer selon son expertise et le contexte reçu. Ce pattern est optimisé pour les environnements à faible latence et forte interactivité : assistants conversationnels, support client, interfaces vocales temps réel. Il présente cependant une contrainte sérieuse : la traçabilité et le débogage sont nettement plus difficiles quand aucun acteur central ne maintient la vue d'ensemble du workflow. Il est déconseillé pour les workflows qui requièrent une coordination parallèle synchrone ou dont les responsabilités entre agents se recoupent.

Le **pattern personnalisé** donne aux équipes un contrôle programmatique complet sur la logique d'orchestration, les relations entre agents et les règles d'exécution. Via des SDK comme celui de Kore.ai ou des frameworks comme LangGraph, les développeurs codent les transitions d'état, les gardes et les conditions d'escalade. C'est le pattern des industries fortement réglementées — finance, santé, assurance — où les modèles propriétaires et les

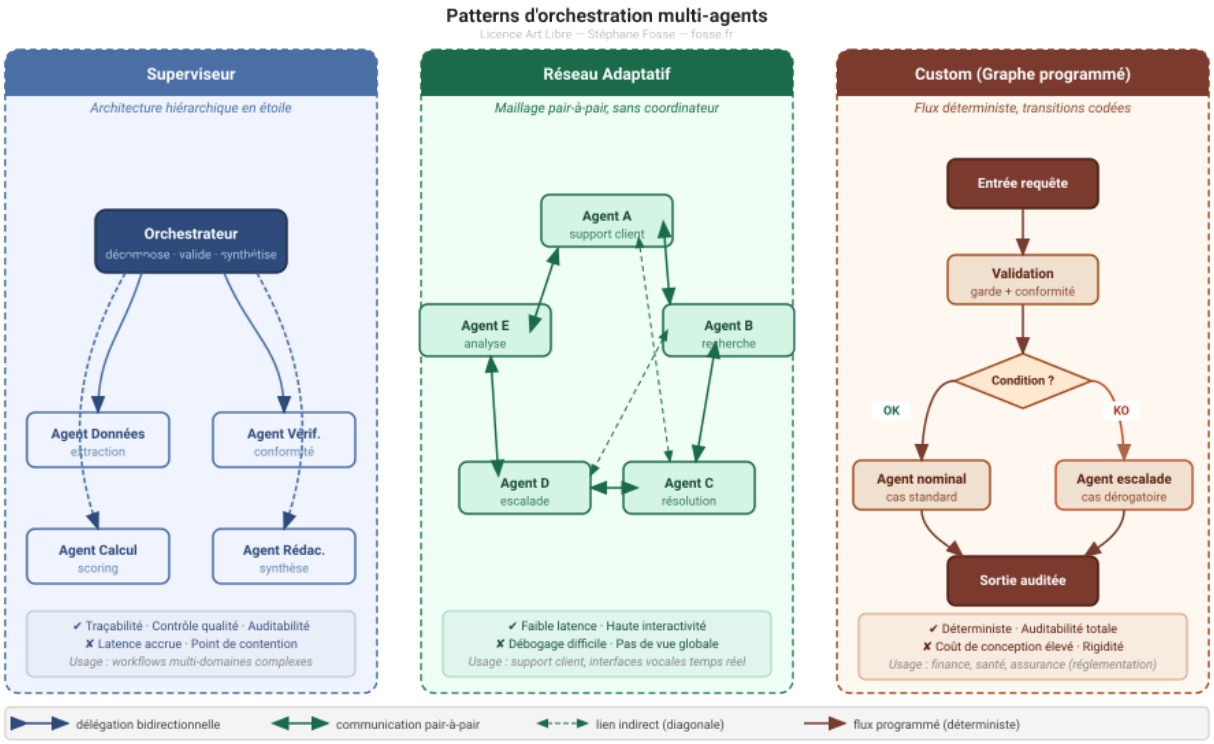

FIGURE 2 – Les trois patterns d'orchestration multi-agents

exigences d'audit rendent les patterns génériques insuffisants. Le coût d'entrée est élevé (conception, maintenance, tests), mais il offre le comportement déterministe que certains contextes imposent.

La recherche académique et les déploiements industriels identifient également des patterns plus granulaires décrits par leur topologie de communication : la chaîne (agents en séquence linéaire, chacun traitant la sortie du précédent), l'étoile (un agent central distribuant vers des agents feuilles sans communication inter-feuilles), et le maillage (communication directe entre toute paire d'agents, pour les problèmes collaboratifs complexes).

Comment s'organisent les rôles dans un système multi-agents ?

Un système multi-agents bien conçu ne consiste pas simplement à multiplier des agents génériques. Il requiert une spécialisation des rôles. Adimulam, Gupta et Kumar [1], dans leur article publié sur arXiv en janvier 2026, identifient trois catégories fonctionnelles.

Les **agents worker** constituent la couche d'exécution. Ils réalisent des tâches précisément délimitées : extraction de données, calcul de scores, génération de brouillons. Certains sont stateless (chaque requête est indépendante), d'autres sont stateful (ils maintiennent un contexte sur plusieurs étapes d'un workflow). Dans un système d'évaluation de risque de crédit, les agents worker extraient les données du dossier, calculent les scores préliminaires, génèrent les projets d'évaluation.

Les **agents service** fournissent des capacités opérationnelles partagées que les autres agents consomment pendant l'exécution du workflow. Ils agissent comme des utilitaires réutilisables : assurance qualité, vérification de conformité, diagnostic, récupération automatique. Un agent de diagnostic inspecte les incohérences dans les données extraites, trace les anomalies au module responsable, génère un rapport structuré. Un agent de guérison peut aller plus loin et relancer l'extraction échouée, ou remettre le workflow dans un état stable.

Les **agents support** opèrent au niveau de supervision et d'analyse. Là où les agents service interviennent dans le flux d'exécution, les agents support maintiennent la santé globale du système : monitoring de la latence de décision, détection de dérive des modèles de risque, analyse des patterns d'approbation, actualisation des datasets. Ils informent l'orchestrateur et les superviseurs humains, sans exécuter directement les tâches métier.

Cette division du travail n'est pas uniquement organisationnelle. Elle a des conséquences directes sur la surface d'attaque (chaque agent n'accède qu'aux données nécessaires à sa fonction), la maintenabilité (les agents peuvent être mis à jour indépendamment) et la scalabilité (les agents worker peuvent être multipliés sous charge sans affecter la couche de supervision).

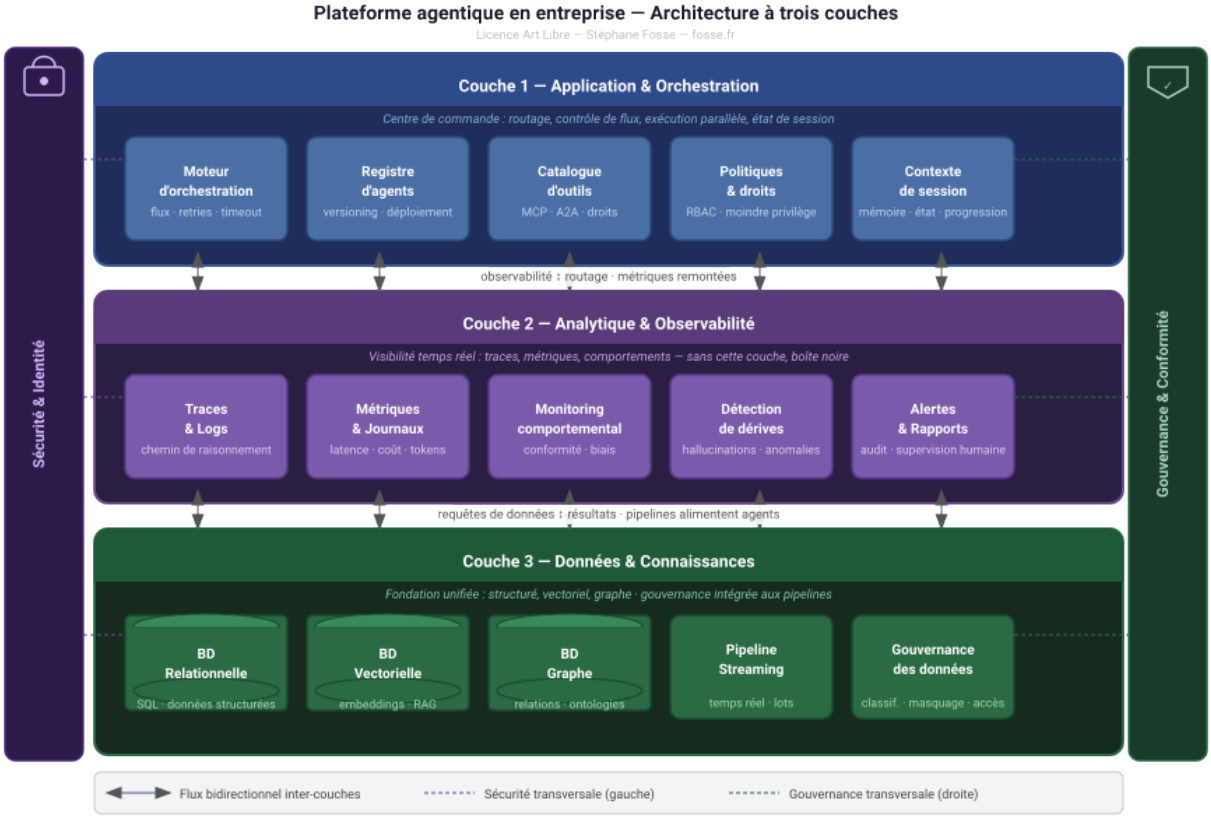

FIGURE 3 – Architecture à trois couches d'une plateforme agentique en entreprise

Quelle est l'architecture d'une plateforme agentique en entreprise ?

Les plateformes IA d'entreprise existantes ont été construites pour des systèmes déterministes : modèles uniques derrière des endpoints API statiques, pipelines ETL figés, gouvernance appliquée après déploiement. Ce design ne tient pas face aux exigences des systèmes agentiques, où plusieurs modèles partagent une mémoire persistante, coordonnent via des workflows multi-étapes, et communiquent directement entre eux via des protocoles standardisés comme le Model Context Protocol (MCP) et Agent-to-Agent (A2A). Les contrôles d'identité hérités supposent que les utilisateurs sont des humains opérant dans des sessions à rôles définis — pas des agents autonomes nécessitant des permissions contextuelles à moindre privilège pour chaque invocation d'outil.

Bain & Company, dans une analyse publiée en avril 2026 [4], décrit la convergence des plateformes agentiques modernes autour d'une architecture à trois couches.

La **couche application et orchestration** est le centre de commande. Elle dirige les workflows multi-étapes via un moteur d'orchestration qui gère le flux de contrôle, les nouvelles tentatives, les délais d'attente et l'exécution parallèle. Chaque requête est routée vers un agent spécialisé via cette couche, qui maintient également le contexte de session partagé, la mémoire d'état et la progression des tâches. Les agents sont déployés comme des services versionnés indépendants, scalables, mis à jour et annulés séparément. Un registre central avec un catalogue d'outils gouverné définit ce qui est disponible, pour qui, avec quels droits.

La **couche analytique et observabilité** fournit une visibilité temps réel sur l'exécution des agents via métriques, journaux et traces collectés à travers les agents, les workflows et l'infrastructure. La traçabilité complète du chemin de raisonnement capture chaque étape, de la requête initiale à l'invocation d'outil jusqu'à la sortie finale. La détection des dérives comportementales, des patterns d'hallucination et des signaux de biais permet aux équipes de maintenir la conformité. Sans cette couche, les systèmes multi-agents deviennent rapidement des boîtes noires.

La **couche données et connaissances** est le fondement de données du système agentique. Elle unifie des données structurées et non structurées via des interfaces standardisées, en couvrant des bases relationnelles, vectorielles et graphes. Des pipelines de streaming temps réel complètent le traitement par lots pour garantir que les agents opèrent sur des données à jour. La gouvernance des données — classification, masquage, rétention, contrôle d'accès inter-domaines — est intégrée aux pipelines plutôt qu'appliquée en bout de chaîne.

Ce qui distingue ce cadre d'une simple description technique, c'est l'insistance des auteurs sur un principe

architectural : sécurité et gouvernance ne sont pas des couches supplémentaires, elles sont transversales aux trois niveaux. Dans les architectures héritées, la gouvernance était appliquée après déploiement. Dans les plateformes agentiques, elle doit être consubstantielle à la conception, car les agents autonomes créent des risques nouveaux — exécution non supervisée, accès multi-systèmes, propagation d'erreurs en cascade — que les contrôles classiques ne sont pas dimensionnés pour gérer.

Quels défis pratiques pose le déploiement en production ?

Les frameworks open source comme LangChain, AutoGen ou CrewAI fonctionnent bien pour les prototypes. À l'échelle de production, plusieurs problèmes émergent qui relèvent moins de l'ingénierie des prompts que de l'architecture des systèmes distribués.

La gestion de la mémoire à long terme est un défi non résolu. La fenêtre de contexte active d'un LLM reste limitée : il faut décider en temps réel quelles informations injecter depuis la mémoire externe, dans quel ordre, avec quelle priorité. Les systèmes actuels combinent généralement des approches de récupération par similarité sémantique (recherche dans des bases vectorielles), de récence et d'importance estimée. La consolidation — le passage d'une mémoire épisodique brute vers des abstractions sémantiques réutilisables — reste un mécanisme expérimental dans la plupart des déploiements.

La coordination entre agents introduit des surcoûts de communication qui peuvent dégrader les performances si le design du système n'en tient pas compte. Les patterns de communication (chaîne, étoile, maillage) ont des implications directes sur la latence, la cohérence de l'état partagé et la résilience aux pannes d'agents individuels. Un système en étoile simplifie la coordination mais crée un point unique de contention ; un maillage distribue la charge mais rend le débogage exponentiellement plus complexe.

La question de l'identité des agents est souvent sous-estimée. Les contrôles d'accès traditionnels, fondés sur des sessions humaines à rôles définis, ne s'appliquent pas directement à des entités non humaines qui invoquent des outils de manière autonome et contextuelle. Il faut des mécanismes de propagation d'identité et de permissions à moindre privilège spécifiquement conçus pour les agents, qui restreignent dynamiquement les droits à la portée strictement nécessaire à chaque invocation.

Enfin, l'évaluation des systèmes agentiques est autrement plus difficile que l'évaluation d'un modèle statique. Les métriques pertinentes incluent coût par décision, latence, précision des actions, stabilité comportementale et résistance aux attaques (prompt injection, manipulation de mémoire). Ces dimensions sont souvent interdépendantes : optimiser la latence en réduisant les tours de raisonnement peut dégrader la précision ; renforcer la gouvernance peut augmenter les coûts en tokens.

Conclusion

Les agents IA fondés sur des LLM ne sont pas une évolution incrémentale des chatbots. Ils représentent un changement de catégorie : de systèmes qui répondent à des systèmes qui agissent. Ce changement exige une relecture sérieuse des habitudes architecturales. La boucle ReAct reste le socle conceptuel de tout agent, mais les architectures de production s'en éloignent rapidement vers des graphes d'orchestration explicites, des registres d'agents gouvernés et des couches d'observabilité temps réel. Les patterns multi-agents — superviseur, réseau adaptatif, custom — ne sont pas interchangeables : chacun a ses conditions d'usage, ses compromis et ses zones de risque propres. La plateforme agentique en entreprise n'est pas une pile technique de plus : c'est une infrastructure pour de la cognition distribuée, avec tout ce que cela implique en matière de gouvernance, de traçabilité et de contrôle. Les protocoles de communication inter-agents (MCP, A2A), les mécanismes de mémoire à long terme et les défis de sécurité associés feront l'objet d'articles dédiés sur ce site.

Références

[1] Apoorva Adimulam, Rajesh Gupta et Sumit Kumar. The Orchestration of Multi-Agent Systems: Architectures, Protocols, and Enterprise Adoption. Anglais. Preprint. arXiv:2601.13671. Skan AI, 2026.

[2] V. Arunkumar, G.R. Gangadharan et Rajkumar Buyya. Agentic Artificial Intelligence: Architectures, Taxonomies, and Evaluation of Large Language Model Agents. Anglais. Preprint. arXiv:2601.12560. Anna University / National Institute of Technology Tiruchirappalli / University of Melbourne, 2026.

[3] Kore.ai. Choosing the right orchestration pattern for multi-agent systems. Anglais. 2026.

[4] Eric Sheng et al. The Three Layers of an Agentic AI Platform. Anglais. Bain & Company, 2026.

[5] Shunyu Yao et al. ReAct: Synergizing Reasoning and Acting in Language Models. Anglais. In : *The Eleventh International Conference on Learning Representations (ICLR 2023)*. arXiv:2210.03629. 2023.

[6] Zeyu Zhang et al. A Survey on the Memory Mechanism of Large Language Model-based Agents. Anglais. In : *ACM Transactions on Information Systems* 43.6 (2025), p. 155.