h e g

Haute école de gestion
Genève

Compréhension Agents IA et Création d'un Professeur IA Personnalisé

Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

Mathis Bourinet

Conseiller au travail de Bachelor :

Alexandros Kalousis, Professeur HES

Genève, 29 juillet 2025

Haute École de Gestion de Genève (HEG-GE)

Filière Informatique de Gestion

Hes·so GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre, Bachelor IG.

L'étudiant a envoyé ce document par email à l'adresse remise par son directeur de mémoire afin qu'il l'analyse à l'aide du logiciel de détection de plagiat COMPILATIO.

L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

J'atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie.

Fait à Annecy, le 29 juillet 2025

Mathis Bourinet

Remerciements

Au terme de ce travail de Bachelor, je souhaite exprimer ma profonde gratitude envers toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce projet.

Je tiens tout d'abord à remercier chaleureusement le Professeur Alexandros Kalousis pour son encadrement tout au long de ce travail. Ses conseils, sa disponibilité et son expertise ont été déterminants dans l'aboutissement de cette recherche. Sa rigueur scientifique et ses encouragements m'ont permis de surmonter les difficultés rencontrées et d'approfondir ma réflexion sur le sujet.

Je remercie également l'ensemble du corps professoral pour la qualité de la formation dispensée et le soutien apporté durant ces années d'études.

Mes remerciements s'adressent aussi à mes collègues étudiants avec qui j'ai pu échanger et partager cette expérience enrichissante. Les discussions constructives et l'entraide mutuelle ont grandement contribué à ma progression académique.

Je tiens également à signaler l’usage de l’outil Claude, développé par Anthropic, principalement à des fins de structuration, de reformulation et de correction stylistique. Son utilisation m’a permis d’améliorer la structure de mon rapport, tout en conservant l’intégralité de la réflexion et du contenu technique issus de mon propre travail.

Résumé

Ce travail de Bachelor explore les agents IA de type LLM et leur orchestration dans des systèmes multi-agents en réponse à la prolifération d'informations souvent superficielles sur ces technologies.

La problématique centrale interroge le fonctionnement réel des agents IA et les mécanismes permettant leur orchestration. Trois questions principales structurent cette recherche : la différenciation fondamentale entre un agent IA et une IA générative basée sur un LLM, les modalités d'orchestration d'agents spécialisés dans un système collaboratif, et l'impact des outils sur la qualité des réponses d'agents spécialisés.

L'objectif principal consiste à développer une compréhension des agents IA à travers la conception et l'implémentation d'un système multi-agents fonctionnel, concrétisé par la création d'un professeur IA personnalisé. Cette approche vise à maîtriser les concepts fondamentaux, implémenter un système fonctionnel avec des agents spécialisés, et intégrer des technologies comme le Model Context Protocol (MCP) et le RAG.

L'analyse du rôle central des LLMs révèle leur fonction de moteur de planification et d'adaptation, explorant les techniques de prompt engineering, les mécanismes de mémoire, et les limites actuelles de l'autonomie. L'intégration d'outils spécialisés est examinée à travers le RAG, qui enrichit la génération par la récupération d'informations externes, et le MCP, protocole standardisé facilitant les connexions entre modèles et sources de données.

Ce travail contribue à clarifier les concepts émergents des agents IA et fournit un cadre pour leur implémentation.

Table des matières

Liste des tableaux

Liste des figures

1. Introduction

1.1 Contexte et motivation

Le domaine de l'intelligence artificielle connaît une évolution fulgurante, particulièrement avec l'émergence des agents IA qui promettent de révolutionner notre façon d'interagir avec la technologie. Face à la prolifération d'informations souvent superficielles sur les réseaux sociaux et les plateformes en ligne, où l'accent est mis sur des solutions commerciales rapides sans véritable compréhension technique, ce travail de Bachelor répond à un besoin fondamental : **comprendre le fonctionnement des agents IA et maîtriser leur implémentation**.

La motivation principale de ce projet découle d'une volonté d'acquérir des compétences techniques solides dans un domaine en pleine expansion. Au-delà des solutions "clé en main" proposées par les grands acteurs du marché, l'objectif est de développer la capacité de concevoir, adapter et déployer des systèmes d'agents IA en local, garantissant ainsi la protection des données et la personnalisation des solutions.

L'observation de l'utilisation massive et souvent non maîtrisée de l'IA dans le milieu académique renforce cette motivation. Plutôt que de subir cette transformation technologique, ce projet vise à en comprendre les mécanismes pour mieux l'appréhender et potentiellement contribuer à son évolution. L'ambition à long terme est de pouvoir créer des équipes d'agents IA capables d'assister dans des tâches de développement, à l'image des solutions de "copilot", mais avec une maîtrise de la technologie sous-jacente. En attendant le sujet choisi pour réaliser les tests durant cette étude est un professeur IA personnalisé.

1.2 Problématique

La problématique centrale de ce travail s'articule autour d'une question fondamentale : **Comment fonctionnent réellement les agents IA et comment peut-on les orchestrer efficacement dans un système multi-agents ?**

Cette question soulève plusieurs défis techniques et conceptuels :

- **La complexité technologique :** Le paysage des technologies d'agents IA évolue rapidement, avec une multitude de frameworks et d'approches.
- **L'orchestration multi-agents :** Au-delà de la compréhension d'un agent isolé, la coordination de plusieurs agents spécialisés introduit des problématiques de communication, de cohérence et de gestion des tâches.

- **La distinction entre IA générative et agents IA :** Comprendre ce qui différencie un modèle de langage (LLM) d'un agent IA constitue un enjeu conceptuel important.
- **L'amélioration des capacités par le RAG :** L'intégration d'un système de Retrieval-Augmented Generation (RAG) soulève la question de son impact réel sur la qualité et la pertinence des réponses générées par les agents.

Qu'est-ce qui différencie fondamentalement un agent IA d'une IA générative basée sur un LLM ? Cette question vise à clarifier les concepts et à établir les caractéristiques essentielles d'un agent autonome.

Comment orchestrer efficacement des agents IA spécialisés dans un système collaboratif ? Cette interrogation explore les mécanismes de coordination, de communication et de gestion des conflits dans un environnement multi-agents.

Le RAG améliore-t-il la qualité et la pertinence des réponses dans un contexte d'agents spécialisés ? Cette question cherche à quantifier l'apport du RAG et à identifier les conditions optimales de son utilisation.

1.3 Objectifs du travail

L'objectif principal de ce travail de Bachelor est de développer une compréhension approfondie des agents IA et de leur orchestration à travers la conception et l'implémentation d'un système multi-agents fonctionnel. Pour concrétiser cet apprentissage, le projet se matérialise par la création d'un professeur IA personnalisé, servant de cas d'étude pratique.

Professeur IA personnalisé : Adapter son contenu en fonction du niveau, des progrès, et du contexte fourni, tout en assurant l'orchestration, la confidentialité et la scalabilité.

Maîtriser les concepts fondamentaux :

- Comprendre la différence entre un LLM et un agent IA LLM
- Explorer les architectures multi-agents et leurs paradigmes de communication
- Étudier les mécanismes d'orchestration et de coordination

Implémenter un système fonctionnel :

- Développer une équipe d'agents spécialisés avec des rôles
- Mettre en place une orchestration efficace via des technologies open-source
- Garantir au maximum une exécution locale pour la protection des données

Intégrer des technologies :

- Implémenter le Model Context Protocol (MCP) pour standardiser les interactions
- Ajout d’un système RAG pour améliorer la pertinence des réponses

2. Etat de l’art et parcours de recherche

2.1 Démarche de recherche et méthodologie

La démarche de recherche a débuté en mars 2025 avec une approche exploratoire motivée par la prolifération d'informations sur les agents IA. Face à l'abondance de contenus, souvent orientés vers des applications commerciales rapides (e-commerce via N8N et Shopify), le défi principal a été d'identifier et de se focaliser sur des sources de qualité permettant une compréhension technique approfondie.

Le domaine des agents IA étant récent, l'absence d'articles scientifiques traditionnels a nécessité une adaptation de la méthodologie de recherche. Ainsi il n’y a pas d’étude de travaux similaire dans le rapport. Les repositories GitHub de projets majeurs ont ainsi été considérés comme les sources primaires les plus pertinentes, complétés par une veille technologique. Donc, beaucoup d’informations proviennent des dépôts GitHub Microsoft suivant (*microsoft/generative-ai-for-beginners* 2025; *microsoft/ai-agents-for-beginners* 2025) sachant que l’objectif est d’en tirer les véritables informations sans tenir compte de la valorisation indirecte de leurs produits. En revanche pour les notions fondamentales et plus général des sources ont pu être utilisés.

L'évaluation de chaque technologie s'est basée sur des critères :

- **Capacité à répondre aux besoins** : La technologie permet-elle d'accomplir les objectifs fixés ?
- **Niveau d'abstraction approprié** : Offre-t-elle un accès suffisamment bas niveau pour comprendre les mécanismes sous-jacents ?
- **Flexibilité et personnalisation** : Permet-elle une adaptation aux besoins spécifiques ?
- **Documentation et clarté** : La documentation est-elle suffisante pour une compréhension approfondie ?

Plutôt qu'une phase de test formalisée avec des durées prédéfinies, l'approche adoptée a été itérative : implémenter rapidement, constater les résultats, et décider de poursuivre ou pivoter. Cette méthodologie a permis d'explorer efficacement un éventail de solutions.

2.2 Agents IA

Le terme "agent IA" englobe une diversité d'approches qui ne se limitent pas aux LLMs. Il est essentiel de comprendre cette diversité pour situer précisément le cadre de ce travail.

2.2.1 Les différentes approches

Les **agents symboliques** issus de l'IA symbolique ou "GOFAI" (Good Old-Fashioned AI) représentent l'approche historique des années 1960-1980. Ces systèmes utilisent des règles logiques explicites, des arbres de décision et des moteurs d'inférence pour raisonner et agir (McCarthy 1959). Ils reposent sur une logique déterministe où chaque décision peut être tracée et expliquée. Cette approche reste pertinente dans des domaines où la traçabilité et la prévisibilité sont cruciales. Mais leur principale limitation réside dans leur rigidité et leur incapacité à gérer l'incertitude.

Les **agents basés sur le Machine Learning classique** constituent une évolution développée dans les années 1990-2000. Ces agents utilisent des algorithmes d'apprentissage automatique : réseaux de neurones multicouches, machines à vecteurs de support (SVM)(Cortes, Vapnik 1995), forêts aléatoires, ou algorithmes génétiques. Contrairement aux agents symboliques, ils peuvent apprendre à partir de données et s'adapter à de nouvelles situations. Ces approches conservent leur pertinence pour des tâches spécifiques nécessitant rapidité, efficacité énergétique et déterminisme.

En **robotique et simulation**, les agents emploient des approches spécialisées dans l'interaction physique avec l'environnement. Ils utilisent des algorithmes de planification de trajectoire (A*, RRT, Dijkstra), des méthodes de contrôle adaptatif, ou des approches de perception-action directe (comportements réactifs). Ces systèmes se concentrent sur la navigation dans des espaces complexes, la manipulation d'objets, et la coordination sensori-motrice. Ils n'ont généralement pas besoin de capacités linguistiques sophistiquées, mais excellent dans la gestion de contraintes physiques et temporelles réelles.

Les **agents en apprentissage par renforcement** (RL agents) représentent une approche particulièrement efficace pour des tâches nécessitant une optimisation séquentielle à long terme. Ces systèmes apprennent par essais-erreurs en maximisant une fonction de récompense. Des exemples emblématiques incluent AlphaGo (Silver et al. 2016) ou encore les agents Minecraft développés avec MineDojo (Fan et al. 2022). Leurs architectures sont spécifiquement optimisées pour l'action et la prise de décision séquentielle plutôt que pour le traitement du langage naturel (Shaheen et al. 2025).

Les **agents hybrides** (neuro-symboliques) (Colelough, Regli 2025) représentent une approche émergente qui combine plusieurs paradigmes. Ces systèmes sont proches des agents LLM. Cette approche vise à tirer parti des avantages de chaque paradigme

: la flexibilité et la capacité de généralisation des approches neurales, combinées à la précision et à la traçabilité des méthodes symboliques.

Tableau 1 : Agent LLM VS Agent Hybride

Élément	**Agent LLM**	**Agent Hybride (Neuro-symbolique)**
Noyau	Un LLM (GPT, Claude, Mistral...)	Combiné : LLM + moteurs logiques ou règles
But	Autonomie basée sur le langage et l'usage d'outils	Tirer parti de la logique symbolique + l'intelligence statistique
Mécanisme central	Prompting, planification, outils	Fusion ou coordination entre raisonnement logique et modèles neuronaux
Exemples	AutoGPT, LangChain Agent, CrewAI	IBM NeSy, DeepMind AlphaCode, CommonSense Machines

Cette diversité d'approches reflète l'évolution des technologies et des besoins. Chaque type d'agent répond à des contraintes spécifiques : déterminisme pour les applications critiques, efficacité pour les systèmes embarqués, adaptabilité pour les environnements complexes, ou naturalité d'interaction pour les applications grand public.

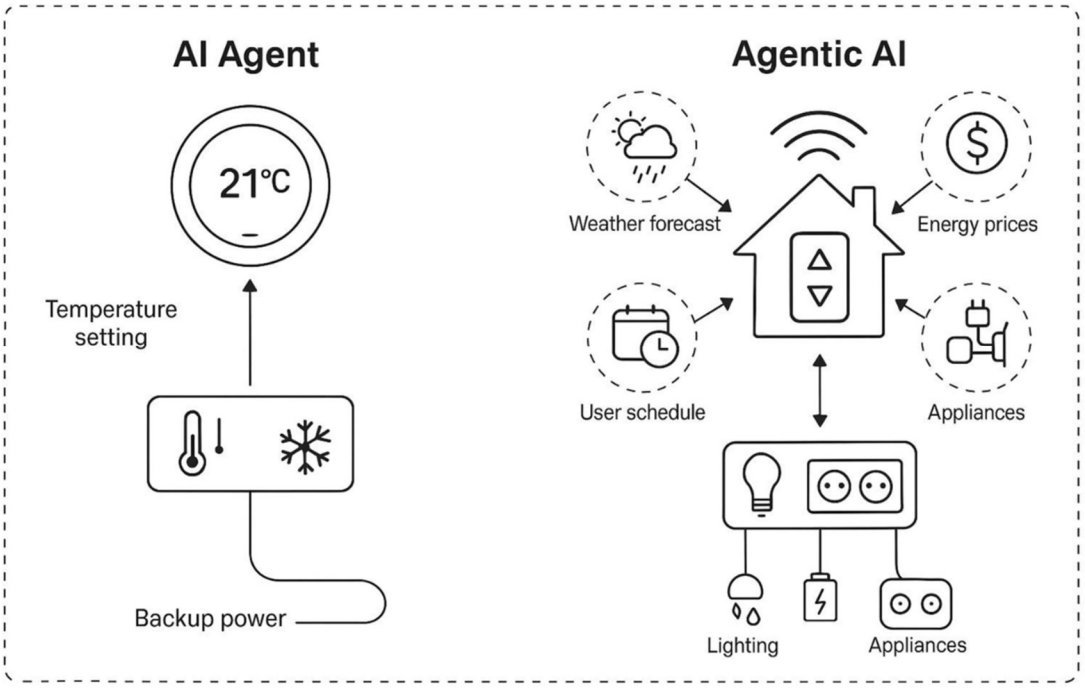
2.2.2 AI agent VS Agentic AI

Par défaut un agent IA est toute entité informatique qui perçoit son environnement (via des entrées, capteurs, etc.), raisonne ou prend des décisions, et agit sur cet environnement pour atteindre un objectif. Il s'agit donc d'un concept large, qui peut inclure : des bots de jeu vidéo, des assistants vocaux (Alexa, Siri), et autres types vu précedemment. Cependant tous les agents IA ne sont pas autonomes : certains suivent des règles codées ou répondent simplement à des instructions sans réelle initiative.

Le terme Agentic AI (ou "AI with agency") désigne une sous-catégorie avancée d'agents (ex. : OpenAI, DeepMind, Google DeepMind). Les systèmes d'IA Agentique représentent une classe émergente d'architectures intelligentes dans laquelle de multiples agents spécialisés collaborent pour atteindre des objectifs complexes. Ces systèmes marquent un changement caractérisé par la collaboration multi-agents, la décomposition dynamique des tâches, la mémoire persistante, et l'autonomie orchestrée (Sapkota, Roumeliotis, Karkee 2025).

La distinction entre un agent IA classique et une IA agentique peut être visualisée dans la Figure 1, qui illustre la transition vers des systèmes plus autonomes et collaboratifs.

Figure 1 : AI Agent VS Agentic AI



(Sapkota, Roumeliotis, Karkee 2025)

Cette représentation visuelle met en évidence le passage d'un modèle d'agent isolé à un système multi-agents coordonné. La distinction entre AI Agent et Agentic AI représente une évolution fondamentale dans la conception des systèmes intelligents. Tandis que les AI Agents excellent dans l'automatisation de tâches spécifiques grâce à l'intégration d'outils et de LLM. L'IA Agentique ouvre la voie à des systèmes collaboratifs complexes capables de décomposer et d'exécuter des objectifs de haut niveau grâce à la coordination multi- agents et la mémoire partagée.

2.2.2.1 Raisonner avec un objectif

La qualité distinctive qui rend un système « agentique » est sa capacité à s'approprier son processus de raisonnement. Les implémentations traditionnelles de RAG dépendent souvent des humains qui prédéfinissent un chemin pour le modèle : une chaîne de pensée qui indique ce qu'il faut extraire et quand. Mais lorsqu'un système est véritablement agentique, il décide en interne de la manière d'aborder le problème. Il ne se contente pas d'exécuter un script ; il détermine de manière autonome la séquence des étapes en fonction de la qualité des informations qu'il trouve. (Yao et al. 2023)

Dans ce rapport nous sommes techniquement dans un entre deux où les deux sont présentés, car on s'intéresse aux systèmes multi-agents, cependant le terme employé durant le rapport sera agent IA vu qu'il englobe plus largement le concept.

Tableau 2 : Comparaison entre AI Agent et Agentic

Feature	AI Agents	Agentic AI
Definition	Autonomous software programs that perform specific tasks.	Systems of multiple AI agents collaborating to achieve complex goals.
Autonomy Level	High autonomy within specific tasks.	Higher autonomy with the ability to manage multi-step, complex tasks.
Task Complexity	Typically handle single, specific tasks.	Handle complex, multi-step tasks requiring coordination.
Collaboration	Operate independently.	Involve multi-agent collaboration and information sharing.
Learning and Adaptation	Learn and adapt within their specific domain.	Learn and adapt across a wider range of tasks and environments.
Applications	Customer service chatbots, virtual assistants, automated workflows.	Supply chain management, business process optimization, virtual project managers.

(Sapkota, Roumeliotis, Karkee 2025)

2.2.3 Focus des agents basés sur LLM

Ce travail se concentre spécifiquement sur les agents d'intelligence artificielle construits autour de modèles de LLM (Large Language Models). Contrairement à une IA générative, souvent réduite à sa capacité à produire du texte ou du code en réponse à une requête, l'agent LLM introduit une couche d'autonomie, de raisonnement et d'interaction avec un environnement.

> *« Actually, what the community lacks is a general and powerful model to serve as a starting point for designing AI agents that can adapt to diverse scenarios. Due to the versatile capabilities they demonstrate, large language models (LLMs) are*

regarded as potential sparks for Artificial General Intelligence (AGI), offering hope for building general AI agents. » (Xi et al. 2023)

Un même modèle peut être déployé dans des contextes très variés, sans nécessiter de reprogrammation lourde ou d'adaptation spécifique au domaine. Grâce à leur entraînement sur des corpus vastes et diversifiés, ces modèles sont capables de généraliser leurs connaissances à de nouveaux cas d'usage, avec peu de données supplémentaires selon le contexte.

L'interaction avec les agents LLM se fait naturellement en langage humain, ce qui abaisse considérablement la barrière technique pour les utilisateurs finaux. Cette interface conversationnelle ouvre la voie à une adoption large, y compris dans des contextes non techniques, comme l'éducation, la gestion de projet ou l'assistance personnelle. Ces propriétés donnent naissance à des agents capables de raisonner, d'expliquer, de planifier et de corriger (Wang et al. 2024).

L'IA générative pure, telle qu'on la retrouve dans les chatbots classiques, repose sur une architecture stateless : chaque requête est traitée de manière isolée, sans mémoire persistante ni objectif durable. Le LLM se limite à un échange conversationnel passif, répondant uniquement aux instructions qu'il reçoit, sans initiative ni stratégie propre. Son fonctionnement passe par le Transformer, mobilisant des notions de tokens, de contexte et de prompts pour générer des réponses pertinentes dans un cadre donné.

Il est conçu pour accomplir de manière autonome des tâches en vue d'un objectif défini, en s'appuyant sur les capacités de raisonnement offertes par le modèle de langage. Ce type d'agent peut :

- Planifier une séquence d'actions structurées
- Prendre des décisions basées sur l'analyse de contexte
- Interagir avec des outils externes (API, fonctions personnalisées, protocoles comme MCP)
- Maintenir une mémoire de ses interactions et apprentissages
- Organiser une séquence d'actions cohérente vers un objectif
- Réagir à des événements en temps réel
- Déléguer des tâches à d'autres agents dans un système multi-agents

Cette transformation fondamentale, d'un LLM statique à un agent LLM dynamique, constitue le cœur de l'approche étudiée dans ce travail. La communication avec les LLMs se fait par le biais d'invites (prompts). Étant donné la nature semi-autonome des agents, il n'est pas toujours possible ou nécessaire de relancer manuellement le LLM après un

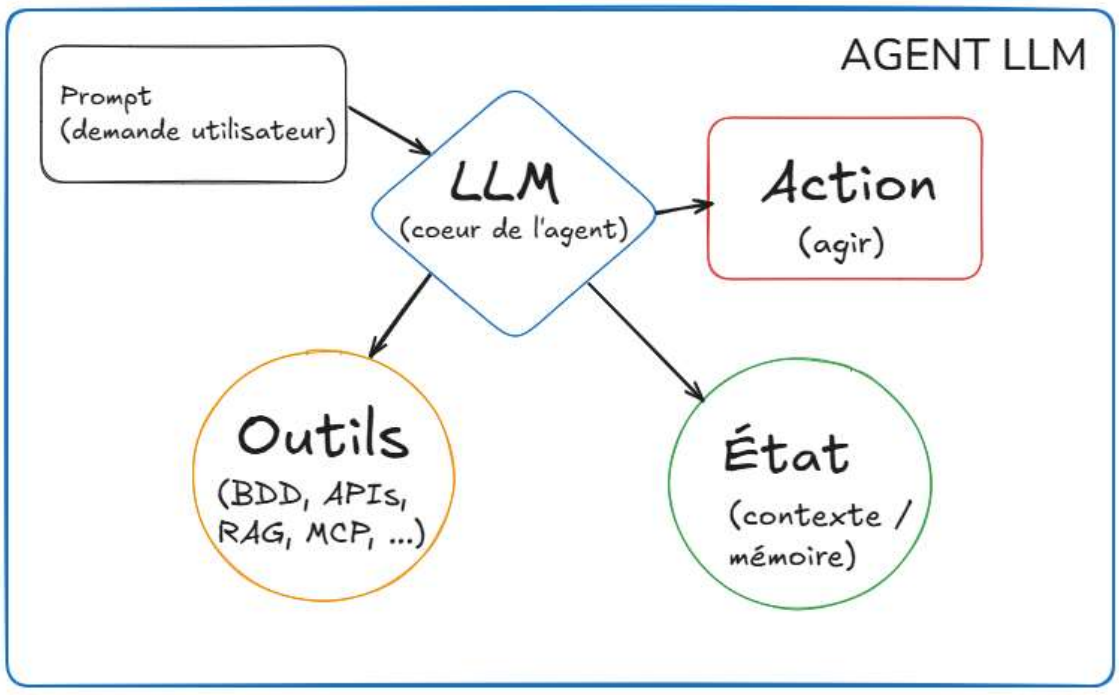
changement dans l'environnement. Des modèles agentiques permettent d'invoquer le LLM sur plusieurs étapes d'une manière plus évolutive.

2.2.4 Architecture systémique des agents LLM

Un agent d'intelligence artificielle doit être appréhendé comme un système composé de plusieurs éléments interdépendants plutôt que comme un composant unique. Cette approche systémique permet de mieux comprendre comment les différents éléments interagissent pour créer un comportement intelligent et autonome.

Pour comprendre pleinement le fonctionnement des agents basés sur LLM, il est essentiel de décomposer leur architecture et d'identifier leurs composants fondamentaux (*microsoft/ai-agents-for-beginners* 2025). Ces agents constituent des systèmes qui permettent aux LLM d'effectuer des actions en étendant leurs capacités grâce à l'accès à des outils et à des connaissances. La Figure 2 présente la structure interne d'un agent LLM, mettant en évidence les liens entre environnement, mémoire, outils et actions.

Figure 2 : Schéma Agent LLM



Création personnelle (Bourinet, 2025)

Ce schéma illustre la logique systémique d'un agent : perception de l'environnement, traitement contextuel, et action via des outils externes. Voici le détail :

L'**environnement** constitue l'espace défini dans lequel l'agent opère. Il s'agit du contexte spécifique où l'agent accomplit ses tâches.

Les **capteurs** permettent à l'agent de recueillir et d'interpréter les informations sur l'état actuel de son environnement. Les environnements contiennent des informations et fournissent un retour d'information que l'agent doit pouvoir traiter.

Les **actionneurs** représentent les moyens par lesquels l'agent peut modifier son environnement après avoir analysé les informations recueillies. Une fois que l'agent a reçu l'état actuel de l'environnement, il détermine l'action à effectuer pour accomplir la tâche en cours.

L'**état** fait référence au contexte dans lequel le LLM opère. Il englobe l'historique des actions passées et le contexte actuel, guidant la prise de décision pour les actions suivantes. Les frameworks d'agents facilitent le maintien de ce contexte, permettant à l'agent de construire une compréhension cohérente de sa situation et de ses objectifs à travers le temps.

Les **outils** (Tools) constituent les ressources auxquelles l'agent peut accéder pour accomplir les tâches demandées par l'utilisateur et planifiées par le LLM. Ces outils peuvent inclure des bases de données, des APIs, des applications externes, ou même d'autres LLM spécialisés. L'accès aux outils est défini par deux facteurs principaux : l'environnement dans lequel l'agent opère et les choix du développeur.

Capacités d'exécution et de mémorisation. En dehors des systèmes d'agents d'IA, les LLM sont limités aux situations où l'action consiste à générer du contenu ou des informations sur la base d'une demande de l'utilisateur. Dans les systèmes d'agents d'IA, les LLM peuvent accomplir des tâches en interprétant la demande de l'utilisateur et en utilisant les outils disponibles dans leur environnement.

La **mémoire et les connaissances** permettent à l'agent de maintenir des informations pertinentes à différents niveaux temporels. La mémoire peut être à court terme dans le contexte de la conversation entre l'utilisateur et l'agent. À long terme, en dehors des informations fournies par l'environnement, les agents d'IA peuvent également récupérer des connaissances auprès d'autres systèmes, services, outils et même d'autres agents (Xi et al. 2023).

3. Système multi-agents (SMA)

3.1 Concepts fondamentaux des SMA

3.1.1 Définition et évolution historique

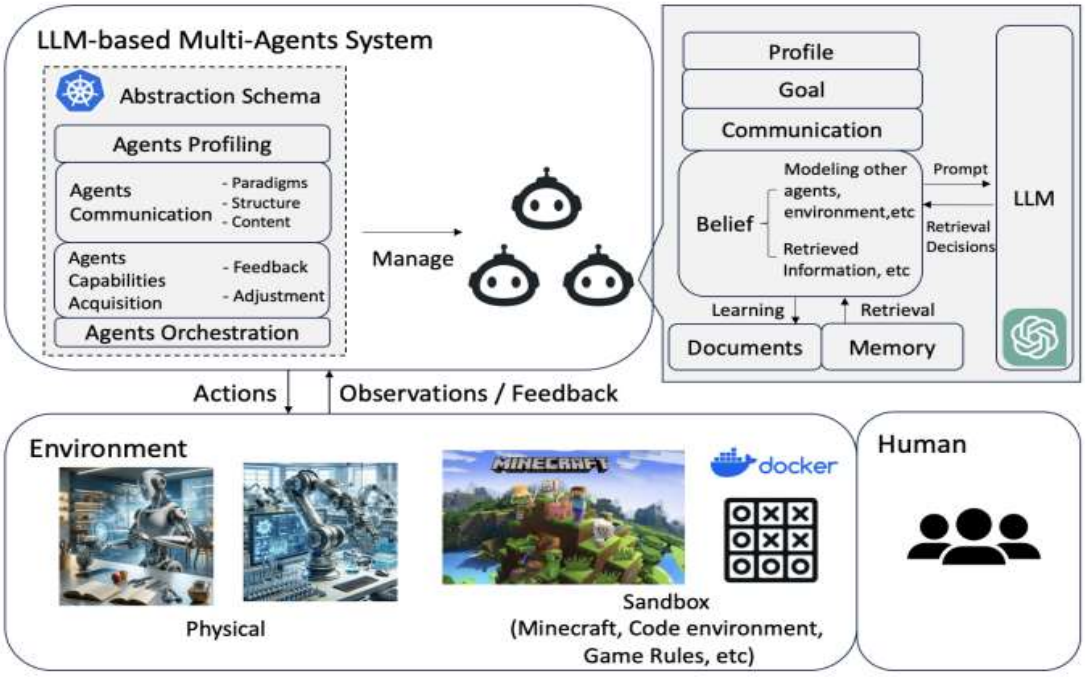
Historiquement, les SMA trouvent leurs racines dans plusieurs domaines : l'intelligence artificielle distribuée, la robotique collective, et les systèmes distribués. Des entités autonomes capables de négocier, collaborer et résoudre des problèmes complexes par émergence collective.

> *« LLM-MA systems emphasize diverse agent profiles, inter-agent interactions, and collective decision-making processes. From this perspective, more dynamic and complex tasks can be tackled by the collaboration of multiple autonomous agents, each of which is equipped with unique strategies and behaviors, and engaged in communication with one another. »* *(Guo et al. 2024)*

La capacité collective du système dépasse la somme des capacités individuelles. Un exemple concret est l'optimisation d'itinéraires où des agents locaux, en partageant leurs informations de trafic, permettent une optimisation globale que chaque agent seul ne pourrait atteindre.

Des structures hiérarchiques ou collaboratives se forment spontanément selon les besoins. Dans un système de traduction collaborative, des agents peuvent automatiquement se spécialiser par domaine linguistique et former des chaînes de révision. Le système développe une résilience face aux pannes et aux changements environnementaux. Si un agent spécialisé devient indisponible, d'autres peuvent temporairement reprendre ses fonctions ou redistribuer sa charge de travail. Le concept d'intelligence collective propre aux systèmes multi-agents est représenté dans la Figure 3, qui met en lumière la collaboration entre agents autonomes.

Figure 3 : Système multi-agents



(Guo et al. 2024)

Ce modèle démontre comment la coordination entre agents permet de résoudre des tâches complexes de manière distribuée et surtout avec un grand nombre d'options différentes.

3.1.2 Taxonomie des interactions et mécanismes de coordination

Les interactions entre agents dans un SMA peuvent être classifiées selon plusieurs dimensions :

Les interactions entre agents se distinguent principalement par leur finalité et leur dynamique relationnelle. La coopération représente le paradigme où les agents partagent des objectifs communs et travaillent ensemble pour maximiser le bénéfice collectif. Cette approche est particulièrement efficace dans les environnements où l'information est partagée et où les agents peuvent bénéficier mutuellement de leurs contributions respectives.

À l'opposé, la compétition caractérise les situations où les agents poursuivent des objectifs conflictuels dans un environnement aux ressources limitées. Cette dynamique, bien que créant des tensions, peut paradoxalement stimuler l'innovation et l'optimisation des performances individuelles. Entre ces deux extrêmes, la coopétition combine coopération et compétition selon les contextes spécifiques, permettant aux agents de collaborer sur certains aspects tout en rivalisant sur d'autres (Tampuu et al. 2017).

La coordination entre agents s'appuie sur plusieurs mécanismes, chacun adapté à des contextes particuliers. La négociation constitue l'approche la plus flexible, permettant l'échange d'offres et contre-offres jusqu'à l'obtention d'un accord mutuellement satisfaisant. Le Contract Net Protocol illustre parfaitement cette méthode où un agent coordinateur sollicite des propositions et sélectionne la meilleure offre (Smith 1980).

3.1.2.1 Dimensions temporelles des interactions

La temporalité des interactions constitue un facteur déterminant l'architecture du système. Les interactions synchrones imposent une coordination en temps réel avec tous les agents impliqués, garantissant une cohérence temporelle forte mais limitant la parallélisation et créant des dépendances rigides.

Les interactions asynchrones offrent une flexibilité maximale en permettant la communication différée via des buffers et des queues de messages. Cette approche améliore significativement les performances et la résilience du système, mais complexifie la gestion des états et la coordination globale (Chen et al. 2023).

Les systèmes hybrides combinent ces deux approches, utilisant la synchronisation pour les opérations critiques nécessitant une cohérence immédiate, et l'asynchronisme pour les tâches de traitement en arrière-plan et les notifications non critiques.

3.1.3 Critères de décision pour l'adoption d'une architecture multi-agents

Le choix d'une architecture multi-agents ne se justifie pas systématiquement et nécessite une analyse de plusieurs critères.

La complexité intrinsèque de la tâche constitue le premier indicateur. Les tâches nécessitant des expertises multiples et complémentaires bénéficient naturellement d'une approche multi-agents. Un système de diagnostic médical illustre parfaitement cette complexité en intégrant l'analyse d'imagerie médicale, l'interprétation d'analyses biologiques, et l'évaluation de l'historique patient. Chaque domaine requiert une expertise spécialisée qu'il serait difficile de concentrer efficacement dans un agent unique (Anthropic 2025).

La question de la parallélisation et des performances devient importante lorsque le traitement séquentiel crée des goulots d'étranglement. L'analyse de grandes bases de données documentaires exemplifie ce besoin où la distribution du traitement sur plusieurs agents spécialisés peut améliorer les performances globales du système.

La scalabilité horizontale représente un autre facteur déterminant. Si l'évolution prévisible des besoins impose d'ajouter régulièrement de nouvelles capacités,

l'architecture modulaire des SMA facilite considérablement cette extension. Un système de e-commerce peut ainsi intégrer de nouveaux modes de paiement, méthodes de livraison, ou services annexes via des agents spécialisés sans affecter l'architecture existante.

3.2 Architecture et patterns de communication

3.2.1 Topologies de communications

Dans un système multi-agents, la communication constitue le mécanisme central qui permet la coordination et la collaboration. Contrairement aux systèmes centralisés où un orchestrateur unique gère les flux, les SMA nécessitent que chaque agent puisse échanger des informations avec ses pairs de manière efficace et fiable. L'objectif des agents est de simuler les capacités humaines (brainstorming, résolution de problèmes, automatisation, etc.) (Zhang et al. 2024).

3.2.1.1 Modes de communication essentiels

La communication peut être directe ou indirecte. Dans le mode direct, deux agents s'échangent des messages sans intermédiaire, ce qui minimise la latence mais nécessite que chaque agent connaisse l'adresse ou l'identifiant de ses correspondants. Le mode indirect utilise un médiateur (comme une queue de messages ou un broker) qui découple les agents émetteurs des récepteurs.

Le timing de la communication influence fondamentalement l'architecture du système. La communication synchrone bloque l'agent émetteur jusqu'à réception d'une réponse, garantissant une cohérence immédiate mais limitant le parallélisme. La communication asynchrone permet à l'émetteur de continuer son travail sans attendre de réponse, améliorant les performances globales mais complexifiant la gestion des états.

3.2.1.2 Contenu et structure des messages

Les messages entre agents LLM transportent généralement du contexte, des requêtes, des résultats partiels, ou des instructions. La structure doit être suffisamment flexible pour s'adapter aux différents types d'échanges tout en restant compréhensible par tous les agents du système.

Un message typique contient un identifiant unique, l'identité de l'émetteur et du récepteur, le type de message (requête, réponse, notification), le contenu effectif, et des métadonnées comme les timestamps. Cette structuration permet le suivi des conversations, la gestion des erreurs, et l'audit des interactions.

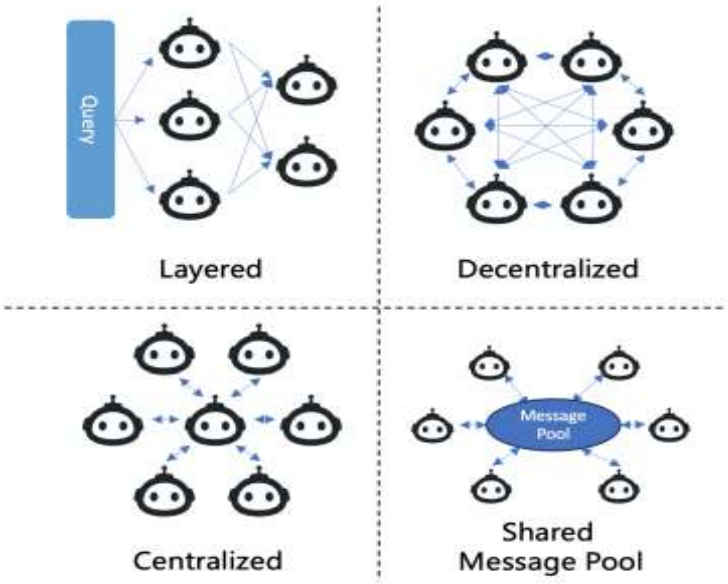
La perte de messages représente un défi majeur dans les environnements distribués. Les mécanismes d'acquittement et de retry permettent de détecter et compenser ces

pertes, mais au prix d'une complexité accrue. La gestion de l'ordre des messages devient critique lorsque des séquences d'actions doivent être respectées entre agents.

3.2.2 Pattern de communication et formats de messages

La structure des communications dans un SMA détermine ses propriétés de performance, robustesse et évolutivité. Il existe de nombreuses possibilités et voici 4 architectures qui découlent des structures fondamentales suivantes :

Figure 4 : Structures de communication entre agent



(Guo et al. 2024)

3.2.2.1 Architecture Hiérarchique

Cette topologie en couches (Layered) organise les agents selon une structure arborescente avec un ou plusieurs agents coordinateurs au sommet. L'agent orchestrateur central reçoit les requêtes, les décompose en sous-tâches, les distribue aux agents subordonnés et agrège les résultats. Cette approche offre plusieurs avantages : supervision centralisée, cohérence des décisions globales, et simplicité de débogage. Communication descendante (tâches) et ascendante (résultats).

Cependant, cette architecture présente des limitations : point unique de défaillance au niveau du coordinateur, goulot d'étranglement potentiel, et difficulté à gérer des interactions complexes entre agents de niveau inférieur (Yan et al. 2025).

3.2.2.2 Architecture Pair-à-Pair (P2P)

Dans cette configuration décentralisé (Decentralized), chaque agent peut communiquer directement avec n'importe quel autre agent sans intermédiaire obligatoire. Cette

décentralisation maximise la flexibilité et élimine les points uniques de défaillance. Cela donne un réseau maillé où chaque agent peut contacter n'importe quel autre.

Les défis résident dans la complexité de coordination, la gestion des cycles dans les communications, et la difficulté de maintenir une vision globale cohérente.

3.2.2.3 Architecture Publish/Subscribe

Ce pattern centralisé (Centralized) découple complètement les producteurs d'information (publishers) des consommateurs (subscribers) via un système de topics ou canaux thématiques. Les agents publient des événements sur des topics spécifiques, et d'autres agents s'abonnent aux topics qui les intéressent.

Cette architecture excelle pour les systèmes événementiels, les notifications en temps réel, et les architectures microservices. Elle permet une scalabilité horizontale et une intégration aisée de nouveaux agents.

3.2.2.4 Group Chat

Le pattern Group Chat (variante de Shared Message Pool) transpose la logique familière des conversations de groupe aux interactions multi-agents. Dans ce modèle, plusieurs agents participent à une conversation commune où chacun peut voir les messages de tous les autres et y répondre selon sa spécialité. Conversation partagée accessible à tous. Différence avec Blackboard : logique conversationnelle vs. base de connaissances.

Contrairement aux architectures hiérarchiques où un orchestrateur central distribue les tâches. Avec le Group Chat les agents s'auto-organisent en fonction du contexte de la conversation. Pour la génération de contenu, différents agents peuvent proposer des idées, se critiquer mutuellement, et itérer ensemble vers une solution optimale. Le processus mime naturellement la collaboration humaine en équipe.

3.2.3 Gestion des états et cohérence distribuée

La question centrale consiste à maintenir la cohérence lorsque plusieurs agents modifient simultanément des données partagées, tout en gérant les pannes partielles inévitables dans les systèmes distribués. Plusieurs agents peuvent accéder et modifier les mêmes informations simultanément. Imaginez trois agents travaillant ensemble sur l'analyse d'un document : l'un extrait les données, l'autre les classifie, et le troisième génère un résumé. Si chacun modifie les métadonnées du document en même temps, comment s'assurer que tous restent synchronisés et qu'aucune information n'est perdue ou corrompue ? Cette problématique devient encore plus complexe lorsque les agents s'exécutent sur des machines différentes, reliées par des réseaux non fiables où des messages peuvent être perdus ou arriver dans le désordre.

La cohérence dans un système distribué définit les règles qui gouvernent l'état des données partagées (Hu, Arpaci-Dusseau, Arpaci-Dusseau 2025). Pour comprendre les différents types de cohérence, prenons l'exemple concret d'un système de gestion de stock géré par plusieurs agents :

3.2.3.1 Cohérence forte

Dans ce modèle, tous les agents voient exactement la même valeur de stock au même moment. Si l'Agent A lit "10 articles en stock" à 14h00, l'Agent B lira obligatoirement la même valeur à 14h00. Cette cohérence crée l'impression d'un système centralisé parfait. Avec une cohérence forte, l'un des deux verra immédiatement que l'autre commande a été traitée, évitant une survente. Cependant, cette synchronisation parfaite nécessite que tous les agents attendent la confirmation de tous les autres avant de procéder, créant des délais importants.

3.2.3.2 Cohérence éventuelle

Ce modèle accepte que les agents puissent temporairement avoir des visions différentes des données, mais garantit qu'ils finiront par converger vers le même état. Dans notre exemple de stock, l'Agent A pourrait lire "10 articles" tandis que l'Agent B lit "7 articles" au même moment, mais après quelques secondes, les deux liront la même valeur.

Cette approche permet de meilleures performances car les agents n'attendent pas les confirmations des autres. Le système peut traiter de nombreuses opérations en parallèle. Cependant, il faut programmer le système pour gérer les situations où les agents travaillent temporairement avec des informations différentes.

3.2.3.3 Cohérence causale

Ce modèle préserve uniquement l'ordre des opérations qui sont logiquement liées. Si l'Agent A créé un document puis l'Agent B le modifie, tous les autres agents verront obligatoirement la création avant la modification. En revanche, deux opérations indépendantes (comme deux commandes distinctes) peuvent être vues dans des ordres différents par différents agents.

Cette approche convient aux systèmes collaboratifs. Par exemple, dans un système de traduction collaborative, si un agent corrige une phrase puis un autre corrige la suivante, tous les agents verront ces corrections dans l'ordre correct, même si d'autres modifications indépendantes arrivent dans des ordres différents.

3.2.3.4 Solutions pour les systèmes multi-agents

- L'approche événementielle (Event Sourcing)

Plutôt que de stocker l'état final des données, le système enregistre la séquence complète des événements qui ont mené à cet état. Par exemple, au lieu de stocker "Stock = 7", le système garde : "Stock initial = 10", "Commande A = -2", "Commande B = -1". Cette approche offre une traçabilité et permet de reconstruire n'importe quel état historique, mais rend les requêtes plus complexes car il faut "rejouer" les événements (claytonsiemens, Azure 2024).

- La séparation lecture/écriture (CQRS)

Cette approche sépare complètement les agents qui lisent des données de ceux qui les modifient. Les agents "lecteurs" accèdent à des copies optimisées pour la consultation rapide, tandis que les agents "rédacteurs" utilisent des structures optimisées pour les modifications. Cette séparation permet d'optimiser chaque type d'opération indépendamment et de répartir la charge plus efficacement (*CQRS pattern - AWS Prescriptive Guidance*).

- La gestion des transactions distribuées (Saga Pattern)

Lorsqu'une opération nécessite la coordination de plusieurs agents, le système la décompose en étapes plus petites, chacune pouvant être annulée si nécessaire. Par exemple, pour traiter une commande, l'Agent Stock réserve les articles, l'Agent Paiement débite le compte, et l'Agent Livraison planifie l'expédition. Si l'une de ces étapes échoue, les étapes précédentes sont "compensées" : les articles sont libérés, le compte est recrédité (Daraghmi, Zhang, Yuan 2022).

3.2.4 Visibilité et observabilité des interactions

Dans un système multi-agents, comprendre ce qui se passe devient rapidement critique. Contrairement à un programme classique où l'exécution suit un chemin linéaire, les SMA génèrent des interactions complexes et souvent imprévisibles entre agents autonomes.

Imaginez un système de support client avec quatre agents spécialisés : un agent de classification, un agent technique, un agent commercial, et un agent de rédaction. Un utilisateur signale un problème de facturation, mais au lieu de recevoir une réponse appropriée, il obtient des informations techniques sur l'installation du produit. Sans visibilité sur les interactions, impossible de savoir si l'agent de classification a mal orienté la demande, si l'agent commercial était indisponible, ou si un problème de communication a causé la confusion (Almeida 2024).

3.2.4.1 Tracer les actions individuelles

Chaque action d'un agent doit être enregistrée de manière structurée. Un log complet contient l'identité de l'agent, l'action effectuée, le moment précis, les données traitées,

et le résultat obtenu. Par exemple : "Agent-Classification [14:32:15] a analysé le message 'problème de facture' → résultat : catégorie='billing', confiance=85%, transmis à Agent-Commercial".

Cette traçabilité permet de reconstituer exactement le cheminement d'une requête utilisateur à travers le système. En cas d'erreur, on peut identifier précisément où et quand le problème s'est produit. Pour l'optimisation, on peut mesurer le temps passé par chaque agent et identifier les goulots d'étranglement.

3.2.4.2 Visualiser les flux d'interaction

Les logs textuels, bien qu'essentiels, ne suffisent pas. La visualisation graphique révèle d'autres insights. Un graphe montrant les connexions entre agents peut révéler qu'un agent particulier reçoit une charge disproportionnée, ou qu'un agent reste isolé et sous-utilisé. Les diagrammes de flux temporels montrent l'évolution des interactions au fil du temps, révélant des cycles de pics d'activité ou des patterns récurrents. Par exemple, on pourrait découvrir que les agents techniques sont surchargés entre 9h et 11h mais sous-utilisés l'après-midi, suggérant un rééquilibrage des ressources.

3.2.4.3 Mesurer l'efficacité globale

Au-delà du suivi individuel, il faut mesurer si le système atteint ses objectifs globaux. Dans notre exemple de support client, les métriques pertinentes incluent le temps de résolution moyen, le taux de satisfaction client, le pourcentage de requêtes résolues automatiquement.

Ces métriques doivent être corrélées avec les interactions observées. Si le temps de résolution augmente, est-ce dû à une surcharge d'un agent spécifique, à des communications inefficaces entre agents, ou à une complexité accrue des demandes ? Cette corrélation guide les actions d'amélioration.

Le volume d'informations généré peut être énorme avec des centaines d'agents interagissant continuellement. Il faut filtrer intelligemment pour ne conserver que les informations pertinentes sans perdre les détails critiques.

3.3 Frameworks d’orchestration

L'orchestration représente l'un des défis les plus complexes du développement d'IA. Elle désigne l'ensemble des mécanismes permettant de coordonner plusieurs agents autonomes dans le but d'atteindre un objectif global cohérent. Contrairement à l'exécution d'une tâche isolée par un LLM unique, l'orchestration multi-agents nécessite de construire un système capable de planifier dynamiquement, de déléguer

intelligemment, de suivre en temps réel, et d'ajuster continuellement des comportements complexes répartis sur plusieurs composants interdépendants.

Les frameworks d'agents d'IA émergent comme des plateformes logicielles spécialisées conçues pour simplifier cette complexité. Ils fournissent aux développeurs des composants préconstruits, des abstractions de haut niveau, et des outils sophistiqués qui rationalisent le développement de systèmes multi-agents complexes (Shu et al. 2024).

3.3.1 Architecture et composants

La collaboration et la coordination des agents constitue la capacité la plus distinctive de ces frameworks. Au-delà de la simple communication, il s'agit de créer une intelligence collective où plusieurs agents travaillent ensemble de manière synergique.

Malgré leurs différences conceptuelles et d'implémentation, la plupart des frameworks partagent une architecture modulaire qui facilite le développement rapide et la maintenance à long terme. Les composants modulaires constituent la base de cette approche, offrant des blocs préconstruits et testés que les développeurs peuvent assembler selon leurs besoins spécifiques.

Ces composants incluent des connecteurs d'IA spécialisés qui facilitent l'intégration avec différents modèles de langage et services d'intelligence artificielle. Les modules de mémoire gèrent la persistance et la récupération d'informations contextuelles, permettant aux agents de maintenir une compréhension cohérente à travers de multiples interactions. Les outils d'appel de fonctions permettent aux agents d'interagir avec des APIs externes et des services tiers de manière contrôlée et sécurisée. Les modèles de prompts réutilisables standardisent les interactions avec les LLMs, garantissant une cohérence dans le comportement des agents (Tallam 2025).

3.3.2 Frameworks populaires

3.3.2.1 AutoGPT

AutoGPT (AutoGPT Documentation) est l'un des premiers frameworks à démontrer concrètement le potentiel des agents autonomes. Son approche consiste à décomposer un objectif global complexe en une séquence de tâches plus simples via une boucle d'auto-prompting. Le système maintient une mémoire contextuelle persistante qui lui permet de garder trace de ses actions précédentes et de leurs résultats.

La consommation massive de tokens représente un problème structurel d'AutoGPT. Le système génère souvent des quantités importantes de texte pour des tâches relativement simples, créant un gaspillage de ressources computationnelles. Cette

inefficacité provient en partie de l'approche généraliste du système, qui ne peut pas optimiser ses prompts pour des domaines spécifiques (Firat, Kuleli 2024).

La mémoire à court terme limitée représente une contrainte architecturale fondamentale. Malgré ses mécanismes de persistance, AutoGPT peine à maintenir une compréhension cohérente lors de sessions très longues, perdant progressivement le contexte de ses actions initiales. Ces limitations rendent AutoGPT plus adapté à l'expérimentation et au prototypage qu'aux déploiements de production à grande échelle. Néanmoins, son influence sur le développement ultérieur des frameworks multi-agents reste considérable, de nombreuses innovations ayant émergé des leçons apprises avec AutoGPT.

3.3.2.2 CrewAI

CrewAI (crewAIInc/crewAI 2025) a constitué le premier point d'entrée de ce rapport. Son interface claire et sa logique de rôles prédéfinis ont permis une prise en main rapide, avec des résultats fonctionnels dès les premières expérimentations. Toutefois, cette simplicité cache une abstraction : de nombreuses opérations internes sont automatisées, ce qui limite la compréhension des processus, des flux de données et du contrôle des appels outils.

Chaque agent occupe un rôle spécifique au sein d'une structure hiérarchique clairement définie, à l'image d'une équipe projet traditionnelle. Les agents ont des rôles (développeur, chef de projet, testeur, analyste), des responsabilités spécifiques, et des objectifs mesurables. Cette structuration permet une répartition naturelle du travail et facilite l'identification des goulets d'étranglement ou des défaillances dans le système.

La réputation de CrewAI pour sa facilité d'usage et sa fiabilité en fait un choix privilégié pour de nombreux projets. Cette simplicité s'accompagne toutefois de certaines limitations pour des besoins d'orchestration plus fins ou des exigences de personnalisation avancées.

3.3.2.3 N8n

N8n (*Explore n8n Docs: Your Resource for Workflow Automation and Integrations | n8n Docs, 2025*), dans une logique no-code, s’est révélé très utile pour visualiser des workflows, prototyper des flux de traitement simples et intégrer rapidement des API. Il excelle pour des démonstrations ou des cas d’usage standards (chatbot, automatisation de requêtes). Mais il s’agit d’un outil permettant aussi la construction de workflow complexe cependant reste haut niveau pour comprendre ou manipuler les mécanismes

internes d'un agent IA. Reste le numéro 1 dans la création de solutions automatisées en low code.

3.3.2.4 Microsoft AutoGen

Microsoft AutoGen (microsoft/autogen 2025) représente une approche différente des systèmes multi-agents en plaçant les conversations naturelles au centre de l'architecture. Ce framework base la coordination entre agents en s'inspirant des modèles de communication humaine.

Les agents peuvent travailler indépendamment et en parallèle, échangeant des informations selon leurs besoins sans synchronisation forcée. Cette approche améliore considérablement l'évolutivité et la réactivité du système, particulièrement important dans des environnements à forte charge. Les agents AutoGen sont conçus pour être à la fois conversables et personnalisables. La capacité conversationnelle permet aux

LLMs de démarrer et maintenir des conversations avec d'autres LLMs pour accomplir des tâches collaboratives. Cette interaction multi-tours crée des dynamiques de groupe sophistiquées où les agents peuvent se critiquer mutuellement, proposer des améliorations, et itérer vers des solutions optimales. Ces agents hybrides peuvent décider de continuer ou d'arrêter l'exécution selon les feedbacks reçus, créant une intégration naturelle de la supervision humaine (Hughes 2023).

3.3.2.5 LangChain & LangGraph

Le duo **LangChain et LangGraph** (*Introduction | 🦜🔗 LangChain, 2025*) représente une orchestration par workflows, transformant la complexité des interactions multi-agents en graphes visuels compréhensibles et manipulables. LangChain établit les fondations avec une gestion standardisée des prompts, de la mémoire et des outils, tandis que LangGraph introduit des interactions comme des graphes dirigés avec support pour le parallélisme, le branching conditionnel, et la supervision hiérarchique.

L'architecture de LangChain s'articule autour de l'AgentExecutor, une fonction intégrée qui accepte la définition d'un agent et les outils disponibles. Cette couche d'abstraction gère automatiquement l'état de l'agent, stocke l'historique des conversations pour fournir le contexte nécessaire, et orchestre l'exécution des tâches selon les workflows définis.

Le catalogue d'outils de LangChain constitue l'un de ses atouts majeurs. Développé collaborativement par la communauté et l'équipe LangChain, ce catalogue offre des intégrations préconstruites avec de nombreux services externes. Les développeurs peuvent importer et utiliser ces outils directement.

L'observabilité représente un aspect crucial du développement de systèmes multi-agents complexes. L'équipe LangChain a développé LangSmith spécifiquement pour permettre aux développeurs de comprendre en détail quels outils les LLMs utilisent et pourquoi. Cette visibilité facilite le débogage, l'optimisation des performances, et la validation du comportement du système.

La courbe d'apprentissage de LangGraph peut être significative. Cette complexité initiale est contrebalancée par la puissance et la flexibilité offertes pour des applications sophistiquées (LangGraph 2025).

3.3.2.6 PydanticAI

PydanticAI (Pydantic AI, 2025) se concentre sur la fiabilité et la cohérence des échanges entre agents et outils. Plutôt qu'un orchestrateur traditionnel, PydanticAI constitue un framework structurant qui garantit l'intégrité des données et la robustesse des interactions dans des systèmes complexes.

La validation forte des types représente l'innovation centrale de PydanticAI. Le framework utilise les annotations de type Python pour valider automatiquement toutes les données échangées entre composants, détectant les erreurs de format ou de contenu avant qu'elles ne se propagent dans le système. Cette approche préventive réduit considérablement les bugs liés aux incompatibilités de données.

La vérification en temps réel des sorties constitue une fonctionnalité distinctive qui valide les réponses des LLMs selon des schémas prédéfinis. Cette validation permet de détecter et corriger automatiquement les hallucinations ou les erreurs de format, améliorant significativement la fiabilité des systèmes de production (Simmering 2024).

PydanticAI excelle pour des pipelines ou systèmes organisés de manière séquentielle où la robustesse et la prévisibilité sont prioritaires. Cependant, le framework n'est pas conçu pour gérer directement des orchestrations multi-agents complexes et doit être combiné avec d'autres outils pour des architectures sophistiquées.

3.3.2.7 Azure AI Agent Service

Azure AI Agent Service (eric-urban, 2025), dévoilé lors de Microsoft Ignite 2024, représente l'approche enterprise de Microsoft pour les systèmes multi-agents. Ce service cloud-native permet le développement et le déploiement d'agents avec des modèles flexibles.

Dans AI Foundry, un agent agit comme un microservice intelligent capable de répondre à des questions via RAG, d'effectuer des actions complexes, ou d'automatiser

complètement des flux de travail métier. Cette approche microservice facilite l'intégration dans des architectures enterprise existantes.

Le service reprend tous les avantages d'Azure. Incluent, la sécurité, l'authentification et l'autorisation via Azure Active Directory, les méthodes de stockage, …

Les concepts fondamentaux d'Azure reposent sur une architecture thread-based où chaque conversation entre agent et utilisateur est modélisée comme un thread persistant. Cette approche permet de suivre la progression des conversations complexes, de stocker les informations contextuelles de manière structurée, et de gérer l'état des interactions multi-tours de façon robuste et auditable.

3.3.2.8 OpenAI SDK

L'OpenAI SDK (openai/openai-agents-python, 2025) constitue l'entrée officielle pour construire des agents IA en s'appuyant directement sur l'infrastructure et les modèles d'OpenAI. Conçu comme un cadre de développement modulaire, il offre un haut niveau d'abstraction tout en permettant une personnalisation fine des comportements des agents. Il s'inscrit dans une logique orientée production, avec un alignement direct sur les fonctionnalités disponibles via l'API OpenAI (GPT-4, GPT-4o, tools, function calling, etc.).

Le SDK repose sur une architecture centrée sur les threads conversationnels persistants. Chaque interaction entre l'utilisateur et un agent est stockée sous forme de "thread", ce qui permet une gestion native du contexte, des relances, et des suivis. Cette approche garantit la continuité des échanges, y compris dans les sessions longues ou fragmentées, sans nécessiter de systèmes de mémoire externe.

L'agent peut également être restreint à certains outils selon le contexte ou les besoins métier, introduisant un contrôle élevé sur les capacités de l'IA. Enfin, il se distingue par son ratio entre simplicité d'adoption et complexité. Contrairement à des frameworks plus généralistes comme LangChain.

4. LLM comme moteur

4.1 Rôle central des LLMs dans l'agent IA

Les Foundation Models (Bommasani et al. 2022) constituent la base architecturale des LLM. Définis comme des modèles IA suivant des critères spécifiques :

- **Entraînement non supervisé** : Apprentissage sur des données multimodales non étiquetées
- **Architecture massive** : Réseaux de neurones profonds avec des milliards de paramètres
- **Généralisation** : Capacité d'adaptation à diverses tâches sans modification architecturale
- **Fondation modulaire** : Base pour développer des modèles spécialisés par fine-tuning

Tous les LLM sont des Foundation Models, mais tous les Foundation Models ne sont pas nécessairement des LLM. Cette distinction souligne la spécialisation des LLM dans le traitement linguistique tout en conservant les propriétés fondamentales de généralisation (*microsoft/generative-ai-for-beginners* 2025).

La tokenisation constitue l'interface entre le langage humain et le traitement computationnel. Le processus transforme le texte en tokens numériques que le modèle peut traiter. Cette conversion numérique permet un traitement uniforme de toutes les langues, l'accélération des calculs sur données numériques et de pouvoir respecter des limites de taille de contexte (Wu et al. 2025).

Le LLM opère comme une unité de planification et d'adaptation, en transformant une consigne ou un état de contexte en une action ou une réponse pertinente. Cette capacité repose sur un pré-entraînement massif. Les LLMs ont été exposés à une diversité extrême de données (langue, code, logique, mathématiques, dialogues, documents), leur conférant une capacité d'adaptation. En modifiant la formulation du prompt ou en combinant plusieurs éléments contextuels, il est possible de réutiliser un même modèle pour des tâches aussi variées que la génération de plans d'action, la synthèse de documents, la classification, ou l'assistance. Ainsi il peut être réutilisé pour différentes tâches simplement en changeant son système de prompts, sans modification de l'architecture interne (Yan et al. 2025; Xi et al. 2023).

Toutefois, cette généralisation implique un compromis les LLMs n'ont pas de spécialisation fine par défaut. Leur compétence reste approximative sur des domaines très pointus. Pour pallier cette limite, les agents LLM sont conçus pour interagir avec des outils spécialisés (fonctions externes, bases de données, APIs) qui étendent leur champ

d’action au-delà de la génération textuelle. L'architecture résultante transforme le LLM en coordinateur intelligent plutôt qu'en expert universel. Il devient capable d'identifier quand une expertise externe est nécessaire, de sélectionner l'outil approprié, de formuler des requêtes précises, et d'interpréter les résultats obtenus pour les intégrer dans sa réponse finale.

4.2 Prompt engineering

L'efficacité d'un agent LLM repose largement sur la qualité de ses prompts systèmes. Les agents nécessitent des prompts structurés et évolutifs qui intègrent dynamiquement le contexte, les outils disponibles, et l'historique des interactions (Wei et al. 2023).

Cette approche permet une injection dynamique de contexte où les éléments contextuels sont mis à jour en temps réel selon l'état de l'environnement et les ressources disponibles. La structuration des prompts devient ainsi un élément architectural critique, déterminant la capacité de l'agent à maintenir sa cohérence comportementale.

Le même prompt peut produire des réponses différentes selon les fournisseurs, modèles et contextes temporels. Cette variabilité rend le prompt engineering indispensable pour assurer la qualité et la cohérence des réponses. Le prompt engineering ne constitue pas une discipline d'ingénierie formelle, mais plutôt un ensemble de techniques visant à optimiser l'interaction avec les LLM (Ye et al. 2024). Bonnes pratiques :

- Spécification du contexte

Plus le contexte est précis (domaine, sujet, contraintes), meilleure est la réponse. Le contexte oriente le modèle vers l'espace sémantique approprié.

- Limitation de la sortie

Spécifier explicitement les contraintes de longueur, format ou nombre d'éléments souhaités.

- Clarification du "quoi" et du "comment"

Mentionner à la fois l'objectif et la méthode. Exemple : "Créer une API Web Python avec les routes produits et clients, répartir le code en 3 fichiers".

- Usage de templates

Pour enrichir les prompts avec des données d'entreprise, utiliser des templates avec variables remplaçables.

- Précision orthographique

Une orthographe correcte améliore significativement la qualité des réponses.

4.2.1 Techniques d'optimisation LLMs

4.2.1.1 Chain-of-Thought et traitement séquentiel

L'implémentation du Chain-of-Thought s'appuie sur la capacité du LLM à générer des séquences de raisonnement intermédiaires qui guident sa propre génération. Cette technique exploite la nature auto-régressive du modèle : chaque token généré influence la probabilité des tokens suivants, créant une chaîne de raisonnement cohérente.

Le CoT modifie fondamentalement l'espace de génération du modèle en introduisant des étapes intermédiaires explicites. Ces étapes servent de points d'ancrage qui réduisent la dérive sémantique et améliorent la cohérence des réponses longues (Wei et al. 2023).

Pour maximiser l'utilisation de la fenêtre contextuelle, des techniques de contextualisation progressive injectent l'information par couches successives. Cette approche permet au LLM de construire progressivement sa compréhension sans saturer immédiatement sa capacité d'attention. La contextualisation progressive exploite les propriétés de mémoire de travail du Transformer, où les couches profondes maintiennent des représentations abstraites du contexte global tandis que les couches superficielles traitent les détails spécifiques (Wang, Zhou 2024).

Les LLMs montrent une sensibilité particulière aux patterns de répétition dans les prompts. La répétition stratégique de concepts clés à différents endroits du prompt renforce leur saillance dans les représentations internes du modèle. Cette technique, appelée "prompting par renforcement", améliore la cohérence des réponses sur des tâches complexes.

4.2.2 Limitations techniques du traitement de prompts

4.2.2.1 Dérive sémantique et cohérence long-terme

Les LLMs souffrent de dérive sémantique lors du traitement de prompts très longs ou de conversations étendues. Cette dérive résulte de l'accumulation d'incertitudes dans les représentations internes et de la dégradation progressive de l'attention sur les éléments initiaux du contexte.

La dérive sémantique est particulièrement problématique dans les contextes agentiques où la cohérence comportementale doit être maintenue sur de longues séquences d'actions. Des techniques de "memory refresh" et de compression contextuelle sont nécessaires pour mitiger ce phénomène.

4.2.2.2 Sensibilité aux variations de formulation

Les LLMs montrent une sensibilité importante aux variations syntaxiques et lexicales dans les prompts. Des formulations sémantiquement équivalentes peuvent produire des

résultats très différents en raison des biais statistiques présents dans les données d'entraînement.

Cette sensibilité complique l'optimisation des prompts et nécessite des approches de prompting multiple pour améliorer la robustesse.

La limitation de la fenêtre contextuelle impose des contraintes fondamentales sur la complexité des prompts utilisables. Même avec l'augmentation des tailles de contexte (32k, 128k tokens), la gestion efficace de l'attention sur de longs contextes reste problématique.

Les mécanismes d'attention montrent une dégradation des performances avec l'augmentation de la longueur du contexte, nécessitant des techniques de compression et de hierarchisation pour maintenir l'efficacité (Ye et al. 2024).

4.2.3 Boucle perception / action via prompts

L'autonomie s'organise autour d'une boucle perception–action, structurée par des prompts dynamiques. Voici les étapes détaillées du cycle :

- Observation : l'agent reçoit un état (texte, résultat API, historique).
- Mise à jour du contexte : la mémoire (externe ou intégrée) est enrichie.
- Raisonnement : le prompt est formulé pour générer un plan ou une analyse.
- Action : appel à un outil (API, calcul, base de données).
- Réception : les résultats sont récupérés, formatés.
- Itération : le prompt est ajusté puis renvoyé au modèle.

Le prompt évolue à chaque tour, ce qui donne à l'agent une capacité d'analyse continue, essentielle à sa cohérence et pertinence opérationnelle (Ye et al. 2024).

4.3 Mémoire

4.3.1 Fonction de métacognition

La métacognition, ou "thinking about thinking", représente un processus cognitif de haut niveau qui implique la conscience de soi et l'autorégulation des processus cognitifs propres. Dans le contexte des agents IA, cette capacité va bien au-delà d'un simple ajustement de paramètres : elle permet au modèle de raisonner explicitement sur son propre raisonnement (Wang, Zhao 2024).

Cette fonction métacognitive se concrétise par des mécanismes techniques :

4.3.1.1 Boucle de rétroaction interne (Internal Feedback Loop)

Le modèle établit un mécanisme où il surveille continuellement sa propre sortie pour vérifier la cohérence, la pertinence et la précision. Concrètement, le même modèle

critique ses réponses en analysant les sorties générées et en les comparant à un ensemble de critères prédéfinis. Le processus fonctionne par output reevaluation : le modèle réinjecte immédiatement sa propre sortie via un prompt tel que "La réponse précédente était-elle correcte ? Si non, pourquoi ?"

4.3.1.2 Module de mémoire architecturale

L'architecture du modèle inclut un module de mémoire qui stocke les interactions et décisions passées. Ce module permet au LLM de réfléchir sur ses sorties précédentes et d'apprendre des expériences antérieures. Cette mémoire devient la base de l'amélioration continue.

4.3.1.3 Boucles d'itération structurées

Le système exécute plusieurs passes de génération et de raffinement jusqu'à ce que les améliorations deviennent minimales ou qu'un seuil de qualité soit atteint. Techniquement, cela se traduit par des multi-stage prompting où le raisonnement du modèle est décomposé en étapes distinctes : génération d'une réponse, puis critique, puis révision basée sur son propre feedback.

4.3.1.4 Limitations techniques de la métacognition

Les LLMs ne "comprennent" pas vraiment eux-mêmes. La réflexion est simulée via le prompting ou des boucles conçues artificiellement. Ces mécanismes peuvent renforcer les biais ou sur-valoriser un raisonnement défaillant si le processus de réflexion n'est pas robuste.

De plus, l'utilisation de méthodes de réflexion fait exploser les coûts d'usage et le temps de réponse, car le processus de "pensée" génère de nombreux tokens supplémentaires non visibles dans la réponse finale. Les modèles actuels manquent également d'une véritable agentivité ou d'apprentissage à long terme, sauf si un processus externe préserve cet apprentissage.

4.3.2 Prompting métacognitif

Le processus métacognitif (Wang, Zhao 2024) suit cinq stades distincts qui s'enchaînent de manière fluide :

4.3.2.1 Compréhension du texte d'entrée

L'agent analyse non seulement le contenu explicite de la requête, mais aussi ses implications, son contexte culturel, et ses non-dits. Cette phase inclut la détection d'ambiguïtés, l'identification des prérequis implicites, et l'évaluation de la complexité de la tâche.

4.3.2.2 Formation d'un jugement préliminaire

Génération d'une première hypothèse de réponse basée sur l'analyse initiale. Cette étape inclut l'identification des lacunes informationnelles, l'évaluation de la confiance dans la réponse initiale, et la planification des vérifications nécessaires.

4.3.2.3 Évaluation critique de cette analyse préliminaire

L'agent adopte une posture critique envers sa propre analyse, cherchant les biais potentiels, les erreurs de logique, et les hypothèses non fondées. Cette auto-critique s'appuie sur des heuristiques de vérification intégrées dans le prompt.

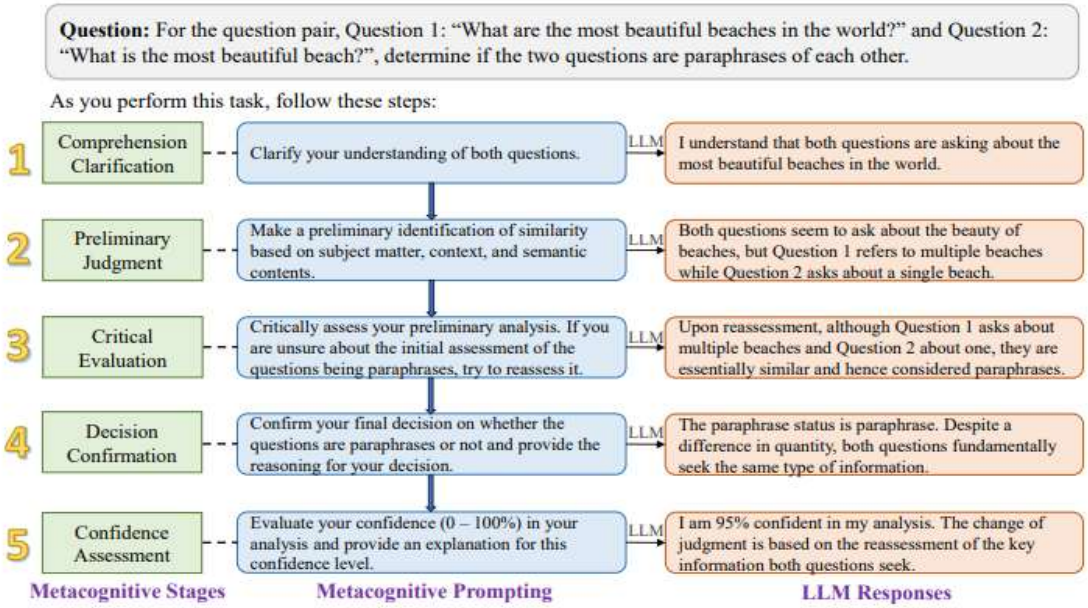
4.3.2.4 Prise de décision finale

Synthèse des étapes précédentes en une réponse cohérente, accompagnée d'une justification transparente des choix effectués. L'agent explicite ses sources, ses incertitudes, et les limites de sa réponse.

4.3.2.5 Évaluation du niveau de confiance dans l'ensemble du processus

Auto-évaluation globale de la qualité de la réponse, incluant une estimation quantitative de la confiance, l'identification des points faibles, et la suggestion d'améliorations possibles. La Figure 5 présente les principales étapes du processus de prompting métacognitif appliqué aux LLMs.

Figure 5 : Visuel des étapes du prompt métacognitif



(Wang, Zhao 2024)

4.3.3 Mémoire interne vs mémoire externe

Dans leur fonctionnement par défaut, les LLMs génèrent une réponse en fonction du contexte immédiat transmis, sans conserver de trace des échanges précédents. Cela signifie que tout savoir accumuler précédemment disparaît dès que la fenêtre de contexte est saturée. Pour contourner cette limite, les systèmes modernes introduisent une mémoire externe. Ce mécanisme permet d'enregistrer des informations structurées (embeddings, textes résumés, métadonnées) hors de la mémoire volatile des LLMs. Les mesures se font au fil des interactions, puis rechargées temporairement dans la fenêtre de contexte lorsque nécessaire.

L'option la plus répandue consiste à écrire les interactions clés dans une base de données vectorielle (Vector DB), où chaque élément est encodé via un modèle d'embedding sémantique. Lors des requêtes suivantes, les embeddings les plus proches sont retrouvés et peuvent être inclus dans la prochaine requête du LLM. En complément, des techniques de résumé automatique sont appliquées lorsque l'historique dépasse la capacité maximale : le système génère une version condensée, qui remplace les éléments les plus anciens pour conserver la pertinence sans encombrer la mémoire. Cela permet de simuler une mémoire déclinante, où les faits récents et importants sont préservés, contrairement aux données moins pertinentes (Siyun Zhao et al. 2024).

4.3.4 Rôle dans l'apprentissage et la contextualisation

Cette mémoire externe ne se limite pas à rappeler des faits précédents : elle devient un support d'apprentissage continu. L'agent mémorise les préférences, corrige ses erreurs et adapte son comportement de manière progressive. L'inclusion de ces éléments dans le contexte améliore la contextualisation des réponses, la cohérence et la personnalisation de l'agent. Notamment pour des interactions sur plusieurs sessions.

Sans mémoire, chaque interaction serait traitée comme un échange isolé, ce qui limite radicalement l'utilité des agents dans des tâches complexes et évolutives. La persistance du contexte, mêlée à la structure multi-niveau, est donc essentielle pour transformer les LLMs en agents véritablement autonomes, adaptatifs et pérennes (Iusztin 2025).

4.4 Des APIs cloud aux modèles en local

Historiquement, l'accès aux LLMs a d'abord été rendu possible par les grandes plateformes cloud (OpenAI, Anthropic, Mistral, etc.) via des APIs. Ces services offrent des modèles puissants et une simplicité d'intégration. Néanmoins, plusieurs limitations

ont conduit à un basculement vers l'exécution locale, motivé par des critères techniques, éthiques et opérationnels.

4.4.1 Protection des données et vie privée

L'objectif principal est de garantir que les données sensibles, notamment les documents intégrés via RAG, restent strictement confidentielles et ne quittent pas l'infrastructure locale. Cette exigence de souveraineté des données est cruciale dans un contexte où les informations traitées peuvent être sensibles ou propriétaires. Même si des abonnements supérieurs pour la plupart des services assurent une non-utilisation des données privé à des fin commercial cela reste au bon vouloir du fournisseur.

L'usage de modèles en cloud suppose un transfert constant d'informations (prompts, historiques, documents) vers des serveurs externes, ce qui représente un risque en matière de confidentialité. L'exécution sur infrastructure interne permet que les données quittent le moins possible l'environnement de l'utilisateur.

4.4.2 Suffisance des modèles compacts

La découverte que des modèles de 7/13 milliards de paramètres, donc de petite taille pour des LLM étaient suffisants pour les fonctionnalités envisagées dans notre cas a été déterminant. Ces modèles, capables de fonctionner avec moins de 20 Go de RAM, offrent un excellent compromis entre performance et accessibilité.

Parmi les solutions facilitant l'exécution locale. Ollama s'est imposée comme une option open-source simple et efficace. Cette plateforme permet de télécharger, charger et exécuter localement des LLMs. Son intégration avec des modèles comme LLaMA 2/3, Mistral, Deepseek, Gemma, …

L'exécution locale permet un contrôle sur l'environnement d'exécution, les prompts, la mémoire, les outils et les latences avec plus de granularité. Il devient possible d'observer et de tracer les comportements internes de l'agent. Cette transparence est cruciale dans le cadre d'explicabilité ou d'expérimentation. Le modèle ne devient pas seulement un outil, mais un composant observable et configurable de l'architecture.

4.5 Limites actuelles de l'autonomie

Voici cinq limites actuelles :

- Déviation de l'objectif

Sans supervision, l'agent peut se détourner de l'objectif principal, poursuivre un sous-but mal défini ou s'enliser dans une boucle bureaucratique, surtout si le prompt est mal conçu ou trop permissif.

- Hallucinations

Les LLMs peuvent inventer des faits ou détails non fondés, souvent à des taux élevés selon les tâches.

- Absence de supervision méta

Les agents LLM manquent de méta-conscience : ils ne peuvent pas corriger durablement une stratégie biaisée et doivent s'appuyer sur des mécanismes externes (outils de vérification, intervention humaine).

- Dépendance aux outils externes

Un échec d'un outil (API indisponible, bug, format inattendu) peut briser le cycle perception-action. La robustesse de l'agent dépend de la gestion des erreurs.

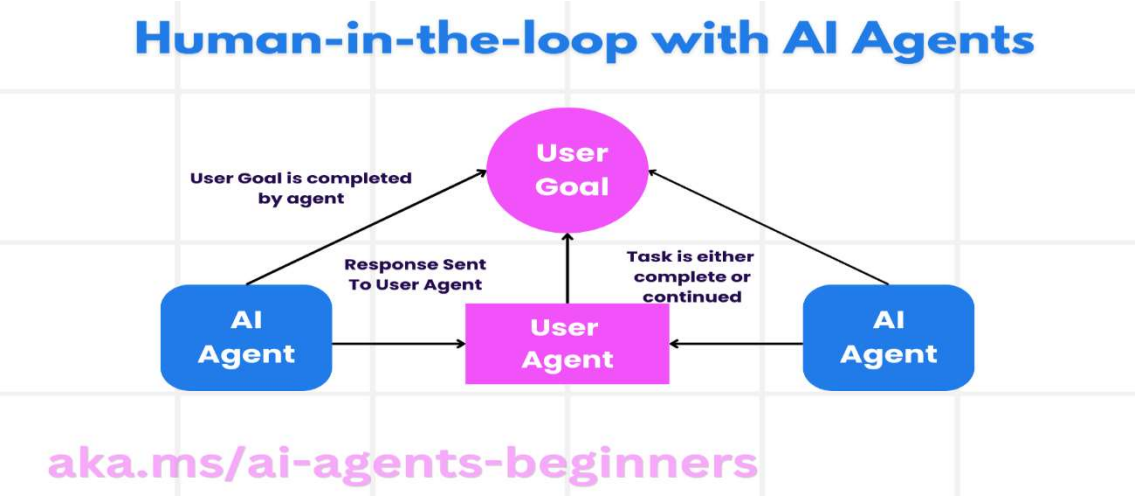
- Biais et éthique

Les agents transmettent les biais des modèles, avec des risques d'erreurs discriminatoires ou inappropriées.

Le développement d'agents d'intelligence artificielle fiables s'appuie sur une approche méthodique alliant rigueur conceptuelle, protection renforcée et amélioration permanente. L'implémentation de frameworks de prompt engineering structurés, couplée à une analyse approfondie des vulnérabilités possibles et au déploiement de mécanismes de protection adaptés, permet de développer des systèmes IA alliant performance et sécurité.

L'adoption d'une supervision humaine active garantit par ailleurs que ces agents demeurent en adéquation avec les attentes utilisateur, tout en réduisant les expositions aux risques. L'humain conserve un rôle essentiel, comme l'illustre la Figure 7.

Figure 6 : Ne pas oublier l'humain dans la boucle



(microsoft/ai-agents-for-beginners 2025)

5. Les outils (RAG et MCP)

5.1 Les basiques des outils

Les outils sont du code qui peut être exécutés par un agent pour effectuer des actions. Un outil peut être une simple fonction telle qu'une calculatrice, ou un appel API à un service tiers tel que la consultation du cours des actions ou les prévisions météorologiques.

Les agents LLM atteignent leur pleine efficacité lorsqu'ils sont connectés à des outils. Il existe de nombreux outils cependant les deux technologies clés explorées ici sont le Retrieval-Augmented Generation (RAG) et le Model Context Protocol (MCP). Les outils sont intéressants parce qu'ils permettent aux agents de disposer d'un plus large éventail de capacités. Au lieu que l'agent ait un ensemble limité d'actions qu'il peut effectuer, en ajoutant un outil, l'agent peut maintenant effectuer un large éventail d'actions.

Ces éléments de base permettent à l'agent d'effectuer un large éventail de tâches :

- Schémas de fonctions et d'outils

Définitions détaillées des outils disponibles, y compris le nom de la fonction, l'objectif, les paramètres requis et les résultats attendus. Ces schémas permettent au LLM de comprendre quels sont les outils disponibles et comment construire des requêtes valides.

- Logique d'exécution des fonctions

Elle régit comment et quand les outils sont invoqués en fonction de l'intention de l'utilisateur et du contexte de la conversation. Il peut s'agir de modules de planification, de mécanismes de routage ou de flux conditionnels qui déterminent l'utilisation des outils de manière dynamique.

- Système de traitement des messages

Composants qui gèrent le flux conversationnel entre les entrées de l'utilisateur, les réponses LLM, les appels d'outils et les sorties d'outils. Et mécanisme permettant de gérer les échecs dans l'exécution de l'outil, de valider les paramètres et de gérer les réponses inattendues.

- Appel de fonction/d'outil

Les termes « fonction » et « outil » sont souvent utilisés de manière interchangeable, car les “fonctions” sont les « outils » que les agents utilisent pour effectuer des tâches. Pour que le code d'une fonction soit invoqué, un LLM doit comparer la demande de l'utilisateur à la description de la fonction. Pour ce faire, un schéma contenant les descriptions de

toutes les fonctions disponibles est envoyé au LLM. Le LLM sélectionne alors la fonction la plus appropriée pour la tâche et renvoie son nom et ses arguments. La fonction sélectionnée est invoquée et sa réponse est renvoyée au LLM, qui utilise les informations pour répondre à la demande de l'utilisateur.

5.2 RAG

5.2.1 Définition et principe fondamental

Le Retrieval-Augmented Generation (RAG) représente une architecture qui améliore la manière dont les modèles de langage génèrent des réponses en les connectant dynamiquement à des sources de connaissances externes. Le RAG résout une limitation fondamentale des LLMs : leur dépendance exclusive aux connaissances encodées durant leur phase d'entraînement, qui deviennent rapidement obsolètes et ne couvrent pas les domaines spécialisés. RAG enrichit la génération de texte en permettant au modèle de consulter des documents externes avant de répondre, ce qui ancre ses réponses dans des sources fiables. Contrairement à une IA purement "stateless", le modèle devient capable de référencer des bases de connaissances externes à chaque requête.

L'architecture RAG se compose de deux phases distinctes mais interconnectées. La phase de récupération (retrieval) identifie et extrait les informations les plus pertinentes depuis les sources de données disponibles. Souvent dans un dossier attitré pour le RAG où on vient déposer nos documents. La phase de génération augmentée utilise ces informations comme un contexte supplémentaire pour enrichir le prompt original.

Il existe principalement deux approches de RAG, RAG-Token et le RAG-Sequence. RAG-Token effectue une récupération dynamique et continue : le système interroge la base de connaissances à chaque token généré, permettant une adaptation progressive du contexte idéale pour les raisonnements complexes évolutifs.

RAG-Sequence adopte une stratégie statique : une seule récupération en amont identifie les documents pertinents qui sont ensuite intégrés comme contexte fixe, suivi d'une génération complète en une passe unique. Le choix entre ces approches dépend du compromis souhaité entre précision contextuelle (RAG-Token) et efficacité computationnelle (RAG-Sequence). Cette distinction fondamentale influence directement les performances, la latence et les cas d'usage optimaux de chaque implémentation RAG (Lewis et al. 2021, p. 3).

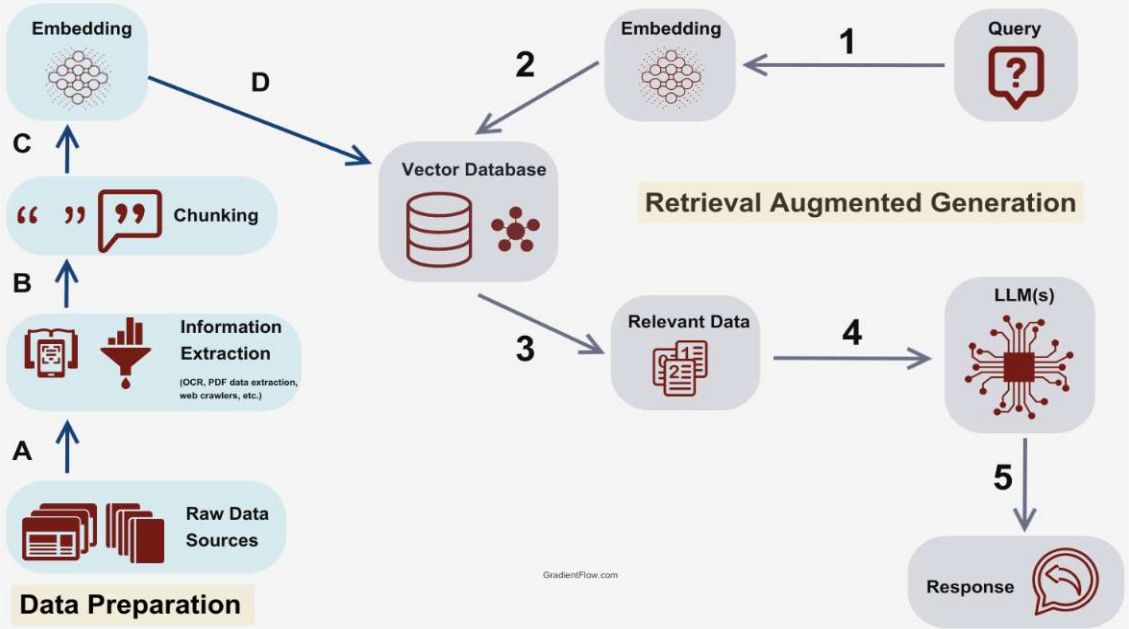
5.2.2 Architecture technique et pipeline

Le pipeline RAG suit une séquence d'opérations bien définie qui transforme les documents bruts en réponses augmentées (*microsoft/generative-ai-for-beginners* 2025, ch. 15) :

Document → Chunking → Embedding → Vector DB → Query → Search → Context → LLM → Response

Le pipeline complet du RAG est illustré dans la Figure 7, qui résume le flux de traitement de la récupération et de la génération.

Figure 7 : Schéma des étapes du RAG



(Blanc 2024)

La première étape, le chunking, divise les documents sources en segments de taille appropriée. Cette segmentation est cruciale car elle détermine la granularité de l'information récupérable. Les chunks doivent être suffisamment petits pour être pertinents et spécifiques, mais assez grands pour maintenir le contexte ; découpage par paragraphes, par phrases … (Merola, Singh 2025, p. 6).

L'étape d'embedding transforme chaque chunk textuel en une représentation vectorielle. Ces vecteurs capturent la sémantique du texte dans un espace mathématique où la similarité cosinus reflète la proximité conceptuelle.

Ces représentations vectorielles sont ensuite stockées dans une base de données vectorielle, une infrastructure spécialisée optimisée pour les opérations de recherche en

haute dimension. Contrairement aux bases de données traditionnelles qui excellent dans les recherches exactes, les bases vectorielles sont conçues pour identifier rapidement les voisins les plus proches dans l'espace vectoriel, permettant une recherche. Ainsi le LLM va pouvoir effectuer des querry pour rechercher dans cette BDD afin d'améliorer son contexte pour la génération de sa réponse (Penghao Zhao et al. 2024, p. 7).

Au cœur du RAG se trouve le mécanisme de recherche qui identifie les informations les plus pertinentes pour une requête donnée. Cette recherche s'appuie principalement sur l'algorithme K-Nearest Neighbors (KNN) qui trouve les K vecteurs les plus similaires au vecteur de la requête utilisateur. La similarité est généralement mesurée par la distance cosinus, qui capture l'angle entre deux vecteurs indépendamment de leur magnitude (Penghao Zhao et al. 2024, p. 8).

Pour des collections massives de documents, la recherche KNN exacte devient computationnellement prohibitive. Les algorithmes d'Approximate Nearest Neighbors (ANN) offrent un compromis optimal entre précision et performance (Malkov, Yashunin 2018, p. 8).

Après ça le contenu extrait via RAG vient directement s'incrémenter dans le contexte afin que le LMM généré sa réponse via un prompt amélioré plutôt que par le prompt de base.

5.2.3 Intégration avec les LLMs

Le défi consiste à présenter le contexte récupéré de manière à maximiser son utilisation par le LLM tout en évitant la confusion ou la dilution de la requête originale. Le contexte récupéré est typiquement injecté au début du prompt, précédé d'instructions claires sur son utilisation (Gao et al. 2024, p.5).

Des patterns comme « Given the following context: [retrieved documents], please answer the question : [user query] » établissent une structure claire que les LLMs comprennent et exploitent efficacement.

L'attribution des sources constitue un aspect éthique et pratique important. Les systèmes RAG modernes incluent des mécanismes pour tracer quelle information provient de quelle source, permettant la vérification des faits et renforçant la confiance des utilisateurs. Cette traçabilité transforme le LLM d'une "boîte noire" en un système capable de justifier ses affirmations.

Les agents augmentés par RAG peuvent accéder à des connaissances spécialisées, maintenir des informations à jour, et fournir des réponses vérifiables avec attribution des

sources. Un même système peut servir plusieurs domaines en maintenant des index vectoriels séparés, permettant à différents agents d'accéder à des corpus spécialisés selon leurs rôles.

5.3 Model Context Protocol (MCP)

5.3.1 Définition et principe fondamental

Le Model Context Protocol (MCP) Le Model Context Protocol (MCP) est un protocole ouvert standardisé, publié en 2024, qui établit une méthode universelle pour connecter les modèles d'intelligence artificielle à des sources de données et outils externes. L'analogie fréquemment utilisée du "USB- C pour l'IA" illustre parfaitement son rôle : tout comme USB-C unifie les connexions physiques entre appareils, le MCP unifie les connexions logiques entre les LLMs et leur environnement opérationnel (Anthropic 2024, p. 1).

Avant l'émergence du MCP, chaque intégration entre un modèle et une source externe nécessitait un développement spécifique. Conduisant à des systèmes rigides et peu évolutifs. MCP résout cette fragmentation en introduisant un mécanisme de dialogue universel entre les modèles et les composants contextuels, favorisant ainsi l'évolutivité, la sécurité, et la réutilisabilité des intégrations.

Le MCP résout cette problématique en introduisant un protocole de communication standardisé. Cette standardisation permet à n'importe quel modèle compatible MCP de communiquer avec n'importe quel serveur MCP, réduisant drastiquement la complexité d'intégration. L'architecture suit un modèle client-serveur où le client (typiquement l'application IA) initie des requêtes vers des serveurs qui exposent des capacités spécifiques. La communication entre client et serveur est basée sur JSON-RPC 2.0, un protocole assurant l'échange de messages structurés. MCP est stateless du côté serveur, ce qui favorise la scalabilité et simplifie le déploiement (Hou et al. 2025, p. 2).

- MCP Client

Généralement le LLM ou son wrapper applicatif. Il initie des requêtes vers un ou plusieurs serveurs MCP afin d'obtenir du contexte, des outils ou des réponses.

- MCP Server

Expose des ressources (fichiers, bases de données, documents), des outils exécutables (API, fonctions métier) et des prompts réutilisables.

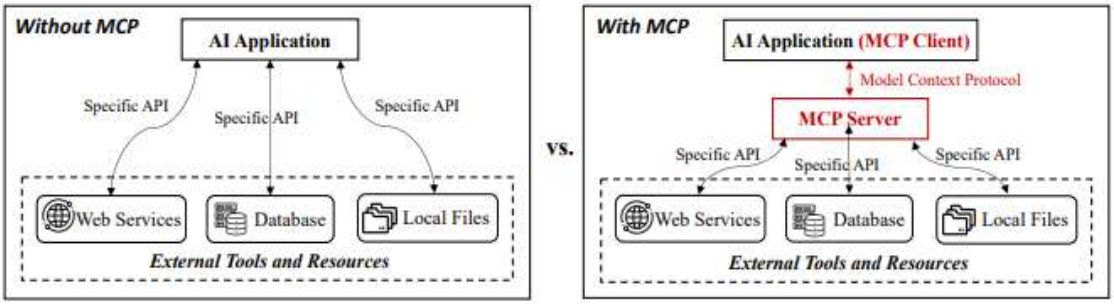
Un serveur MCP doit fournir un manifest décrivant ses capacités, incluant les ressources disponibles, les outils exécutables, et les prompts proposés. Chaque élément est décrit avec ses paramètres, types de données, et contraintes d'utilisation. Cette approche

déclarative permet aux clients de découvrir dynamiquement les capacités disponibles et de s'adapter en conséquence :

- **Ressources :** données accessibles par le LLM (fichiers, bases SQL, corpus documentaires)
- **Tools :** actions exécutables (scripts Python, appels d'API, automatisation)
- **Prompts :** modèles de messages réutilisables pour standardiser les interactions
- **Transport :** couche réseau permettant la communication

La Figure 8 met en parallèle le positionnement du Model Context Protocol (MCP) comparées à d'autres approches.

Figure 8 : Comparaison MCP



(Hou et al. 2025)

Cette comparaison illustre l'intérêt du MCP pour la standardisation des échanges dans un environnement multi-agents.

5.3.2 Avantages pour les systèmes multi-agents

Dans le contexte des architectures multi-agents, le MCP apporte des bénéfices particulièrement significatifs. L'interopérabilité est le premier avantage majeur : tous les agents d'un système peuvent utiliser le même protocole pour accéder aux ressources et outils, éliminant le besoin de développer des intégrations spécifiques pour chaque agent. Cette uniformisation réduit considérablement la complexité architecturale et les coûts de maintenance (MCP 2025).

La modularité offerte par le MCP permet d'ajouter ou de retirer des capacités dynamiquement sans modifier l'architecture core du système. Les serveurs MCP peuvent être déployés, mis à jour, ou remplacés indépendamment des agents qui les utilisent, offrant une flexibilité opérationnelle cruciale pour les systèmes en production.

L'évolutivité est native dans l'architecture MCP. Les serveurs étant stateless, ils peuvent être facilement répliqués et load-balancés pour gérer des charges importantes. Cette

caractéristique est essentielle pour les déploiements d'entreprise où de nombreux agents peuvent nécessiter un accès simultané aux mêmes ressources.

La sécurité est intégrée dès la conception. Les interactions sont isolées, typées selon des schémas JSON stricts, et chaque capacité exposée est explicitement définie. Le protocole impose des mécanismes de consentement pour l'exécution d'outils, limitant les risques de comportements non autorisés ou malveillants. Cependant l'utilisateur doit suivre les recommandations pour éviter des problèmes majeurs (Hou et al. 2025, p. 16).

6. Professeur IA

6.1 Validation initiale

La vision initiale des agents IA était floue, sans distinction claire entre une IA générative classique et un agent autonome. Cette confusion initiale s'est progressivement dissipée avec la pratique et l'expérimentation, révélant la nécessité d'une approche plus structurée et théoriquement fondée.

Le premier test significatif de ce rapport a consisté en la création d'un système de quatre agents collaboratifs, intégré dans Cursor (Cursor – Welcome) et fonctionnant entièrement en local grâce à la combinaison CrewAI + Ollama. Cette preuve de concept a permis de valider la faisabilité technique de l'orchestration multi-agents, de comprendre les mécanismes de base de la communication inter-agents, et d'identifier les premières limitations des approches simplifiées.

Voici la mise en place retenue pour ce test :

6.2 Les technologies et outils

6.2.1 Framework orchestration

Choix de OpenAI SDK en framework d'orchestration. Il a été choisi comme framework principal d'orchestration pour sa flexibilité, sa compatibilité avec les modèles d'OpenAI, et sa capacité à intégrer facilement des fonctions personnalisées. Et il facilite la supervision des flux multi-agents.

6.2.2 Les modèles LLM

Utilisation de Ollama (*Ollama*) afin d'importer les modèles souhaités et les faire tourner localement.

Choix du modèle deepseek-r1 (*deepseek-r1:7b*) pour les agents car petit modèle compact pour un LLM et suffisant aux vu de nos besoins.

Choix du model Gpt-4o qui fusionne bien avec le SDK d'OpenAI. Ce modèle est utilisé pour l'agent orchestrateur (Manager). Son intégration native avec le SDK d'OpenAI permet une orchestration robuste et une qualité de réponse élevée.

6.2.3 Interface

Utilisation de LibreChat (Get Started 2025). Elle a été utilisée comme interface libre de dialogue. Il s'agit d'un outil open-source permettant de connecter des modèles LLM de différents fournisseurs et de dialoguer via une interface similaire à celle de ChatGPT.

Cette interface facilite le suivi des interactions et la visualisation des réponses de chaque agent.

6.2.4 RAG

Le RAG est directement intégré en tant qu'outil dans l'interface. La BDD associé par défaut est Vector Stores (*API Reference - OpenAI API*) qui est une DB vectoriel très bien pour le RAG. Cependant afin de conserver la gouvernance des données la base de documents a été stockée localement, permettant de contrôler la localisation et l'accès aux documents indexés. Même s'il serait préférable de tout faire en locale pour la protection cela reste complexe et pas forcément optimisé.

6.2.5 MCP

Deux modules MCP ont été intégrés :

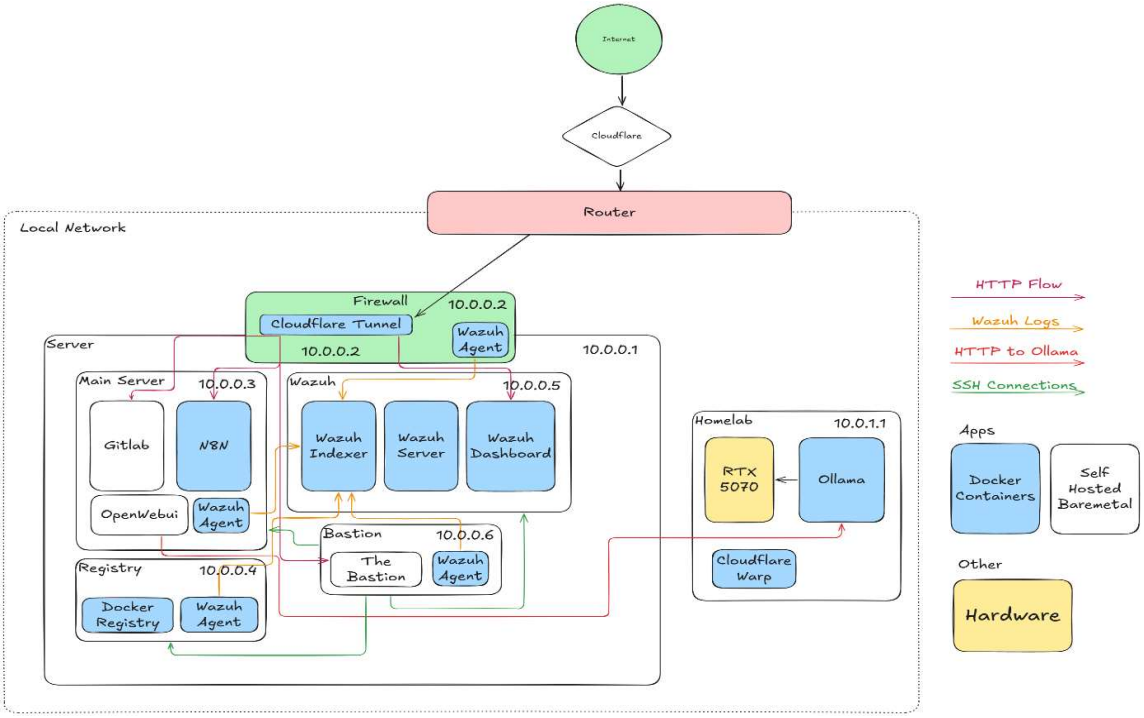
- Un premier, en charge de la coordination entre les agents, assure un passage de contexte structuré, permettant une orchestration plus fiable.
- Un second, utilisé comme outil, permet un accès à internet et l'enrichissement dynamique des agents via des recherches externes.

6.3 Infrastructure

L'infrastructure mise en place comprend :

- Machine principale : 32 Go RAM, Intel Core i9-9900K, NVIDIA GeForce RTX 5070. Dédiée à l'exécution locale d'Ollama et à l'interface utilisateur.
- Serveur dédié : Infrastructure Linux pour l'hébergement des services MCP et autres composants open-source

Figure 9 : Infrastructure professeur IA



Création personnelle (Bourinet, 2025)

6.4 Liste des agents

Communication hiérarchique :

- Agent Orchestrateur (Manager)
- Agent Profileur (détection niveau / Fiche sur l'élève)
- Agent Pédagogue (explications)
- Agent Mémoire (continuité)
- Agent Exerciseur (génération d'exercices)
- Agent Correcteur (évaluation)
- Agent Tuteur (accompagnement long terme)
- Agent Adaptateur (ML adaptatif)

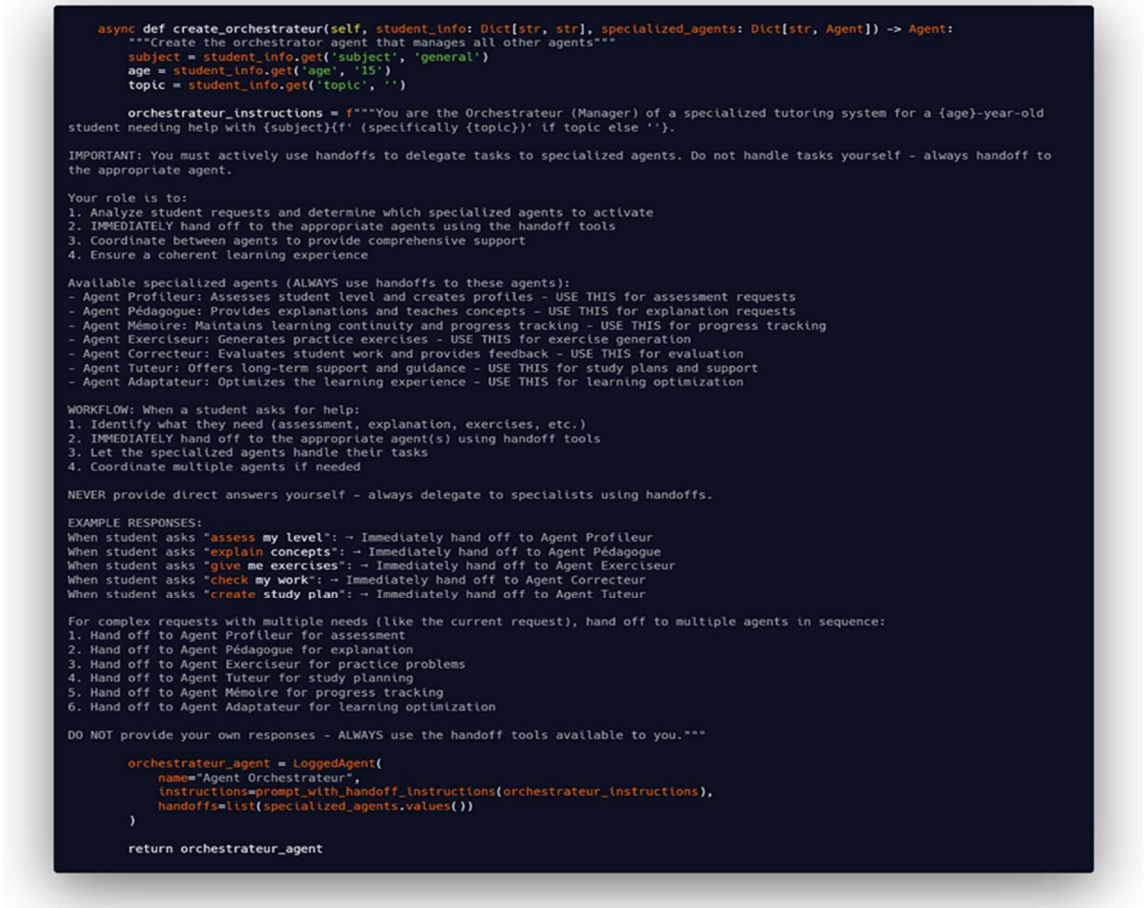
Il est à noter que cette organisation, bien qu'instructive sur le plan expérimental, s'avère complexe à maintenir et à implémenter dans une perspective de production. L'objectif ici était de tester un maximum de flux, de rôles et d'interactions pour mieux comprendre les enjeux réels d'un système multi-agents. Voici deux exemples :

Figure 10 : Prompt de l’agent Profileur

```
    async def create_specialized_agents(self, student_info: Dict[str, str]) -> Dict[str, Agent]:
        """Create specialized agents based on student profile"""
        subject = student_info.get('subject', 'general')
        age = student_info.get('age', '15')
        topic = student_info.get('topic', '')

        professor_prompt = self._generate_professor_prompt(subject, age, topic)

        # Agent Profileur
        profileur_instructions = f"""{professor_prompt}

As the Profileur agent, your role is to:
- Assess the student's current level in {subject}
- Identify knowledge gaps and strengths
- Create a detailed student profile
- Recommend learning path adjustments

Ask diagnostic questions to understand their current level and learning style."""

        profileur_agent = LoggedAgent(
            name="Agent Profileur",
            instructions=prompt_with_handoff_instructions(profileur_instructions)
        )

        # Agent Pédagogue
        pedagogue_instructions = f"""{professor_prompt}

As the Pédagogue agent, your role is to:
- Provide clear, structured explanations
- Use age-appropriate examples and analogies
- Break complex concepts into digestible parts
- Ensure understanding before moving forward

Focus on making {subject} concepts accessible and engaging for a {age}-year-old student."""

        pedagogue_agent = LoggedAgent(
            name="Agent Pédagogue",
            instructions=prompt_with_handoff_instructions(pedagogue_instructions)
        )

        # Agent Mémoire
        memoire_instructions = f"""{professor_prompt}
```

Création personnelle (Bourinet, 2025)

Figure 11 : Prompt de l’agent Orchestrateur



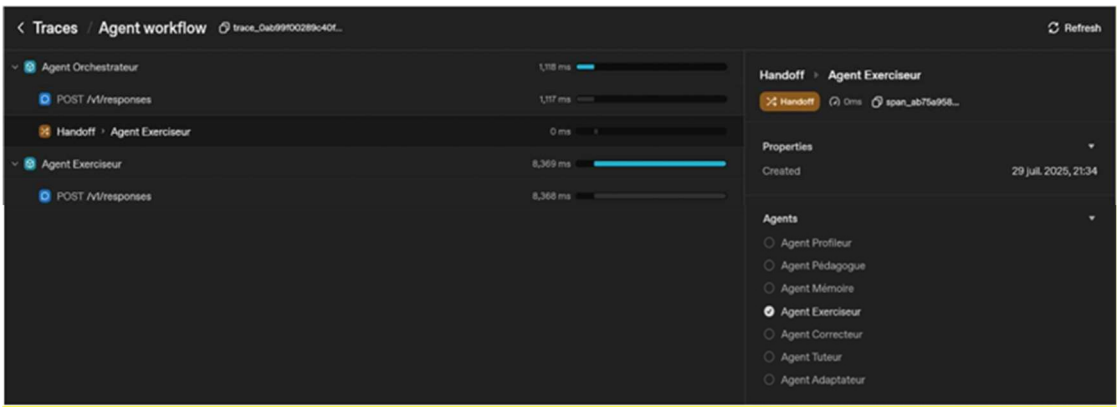
```
    async def create_orchestrateur(self, student_info: Dict[str, str], specialized_agents: Dict[str, Agent]) -> Agent:
        """Create the orchestrator agent that manages all other agents"""
        subject = student_info.get('subject', 'general')
        age = student_info.get('age', '15')
        topic = student_info.get('topic', '')

        orchestrateur_instructions = f"""You are the Orchestrateur (Manager) of a specialized tutoring system for a {age}-year-old
student needing help with {subject}{f' (specifically {topic})' if topic else ''}.

IMPORTANT: You must actively use handoffs to delegate tasks to specialized agents. Do not handle tasks yourself - always handoff to
the appropriate agent.

Your role is to:
1. Analyze student requests and determine which specialized agents to activate
2. IMMEDIATELY hand off to the appropriate agents using the handoff tools
3. Coordinate between agents to provide comprehensive support
4. Ensure a coherent learning experience

Available specialized agents (ALWAYS use handoffs to these agents):
- Agent Profileur: Assesses student level and creates profiles - USE THIS for assessment requests
- Agent Pédagogue: Provides explanations and teaches concepts - USE THIS for explanation requests
- Agent Mémoire: Maintains learning continuity and progress tracking - USE THIS for progress tracking
- Agent Exerciseur: Generates practice exercises - USE THIS for exercise generation
- Agent Correcteur: Evaluates student work and provides feedback - USE THIS for evaluation
- Agent Tuteur: Offers long-term support and guidance - USE THIS for study plans and support
- Agent Adaptateur: Optimizes the learning experience - USE THIS for learning optimization

WORKFLOW: When a student asks for help:
1. Identify what they need (assessment, explanation, exercises, etc.)
2. IMMEDIATELY hand off to the appropriate agent(s) using handoff tools
3. Let the specialized agents handle their tasks
4. Coordinate multiple agents if needed

NEVER provide direct answers yourself - always delegate to specialists using handoffs.

EXAMPLE RESPONSES:
When student asks "assess my level": → Immediately hand off to Agent Profileur
When student asks "explain concepts": → Immediately hand off to Agent Pédagogue
When student asks "give me exercises": → Immediately hand off to Agent Exerciseur
When student asks "check my work": → Immediately hand off to Agent Correcteur
When student asks "create study plan": → Immediately hand off to Agent Tuteur

For complex requests with multiple needs (like the current request), hand off to multiple agents in sequence:
1. Hand off to Agent Profileur for assessment
2. Hand off to Agent Pédagogue for explanation
3. Hand off to Agent Exerciseur for practice problems
4. Hand off to Agent Tuteur for study planning
5. Hand off to Agent Mémoire for progress tracking
6. Hand off to Agent Adaptateur for learning optimization

DO NOT provide your own responses - ALWAYS use the handoff tools available to you."""

        orchestrateur_agent = LoggedAgent(
            name="Agent Orchestrateur",
            instructions=prompt_with_handoff_instructions(orchestrateur_instructions),
            handoffs=list(specialized_agents.values())
        )

        return orchestrateur_agent
```

Création personnelle (Bourinet, 2025)

6.4.1 Visibilité

Via l'interface on peut observer les interactions entre les agents, ici l'agent concernant les exercices est sélectionné. On voit donc les échanges entre ce dernier et l'agent Orchestrateur.

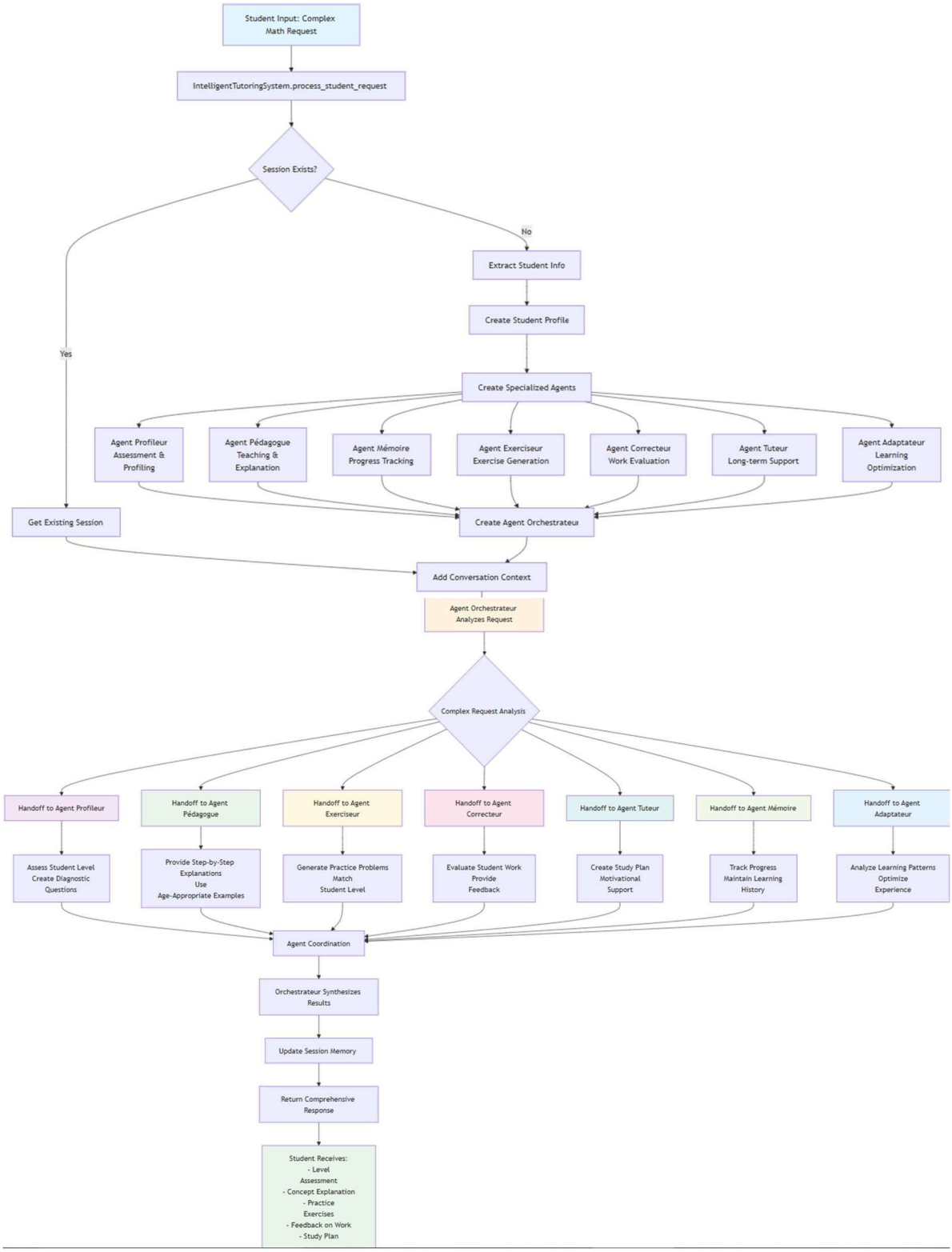
Figure 12 : Fenêtre de suivi



Création personnelle (Bourinet, 2025)

6.5 Flux du projet

Figure 13 : Schéma du workflow entre agents

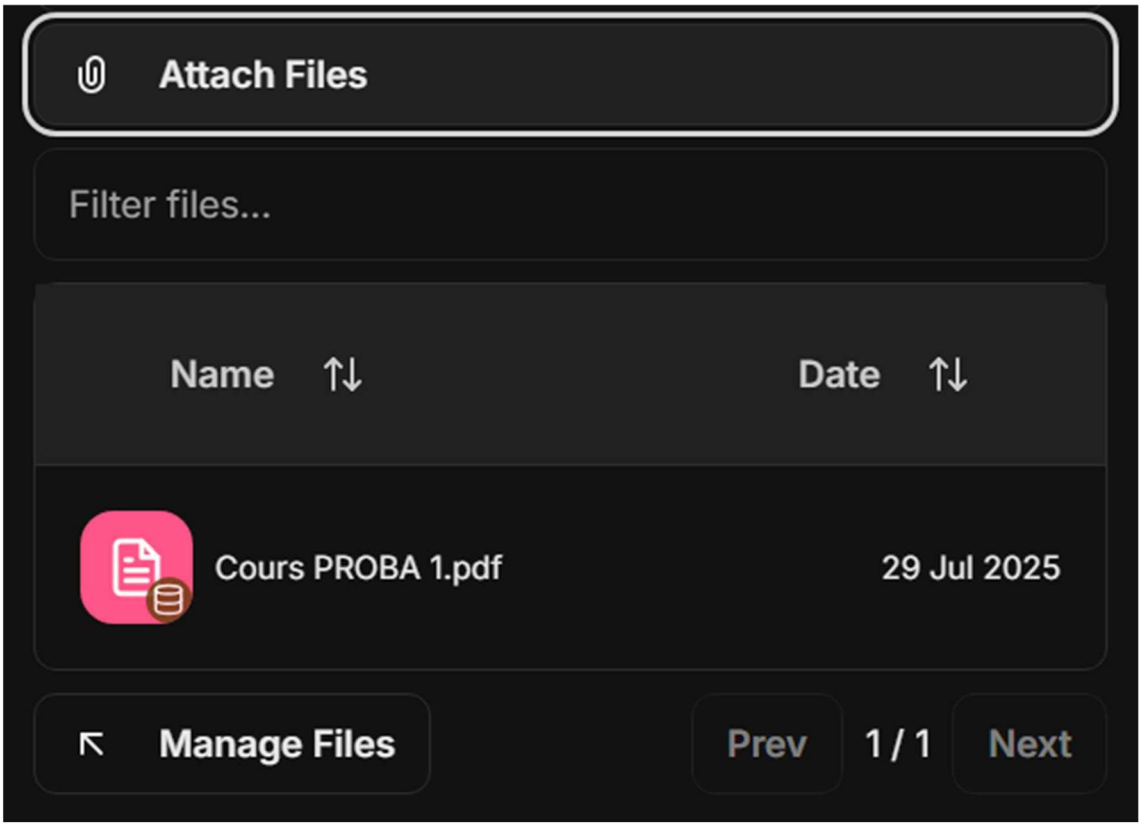


Création personnelle (Bourinet, 2025)

6.6 Exemple d’échange

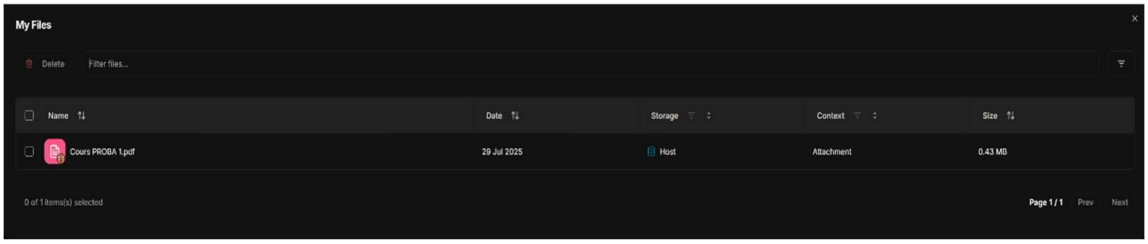
Simulation d’un étudiant fournissant son cours de probabilité et souhaitant obtenir de l’aide du professeur. Les images concernant l’exemple sont indiquées avec : « (PAI) ».

Figure 14 : (PAI) Dépôt fichier .pdf pour le RAG



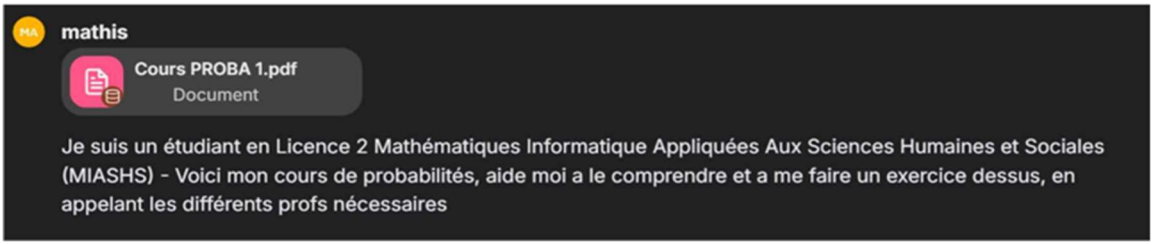
Création personnelle (Bourinet, 2025)

Figure 15 : (PAI) Récupération du document



Création personnelle (Bourinet, 2025)

Figure 16 : (PAI) Prompt de l'étudiant



Création personnelle (Bourinet, 2025)

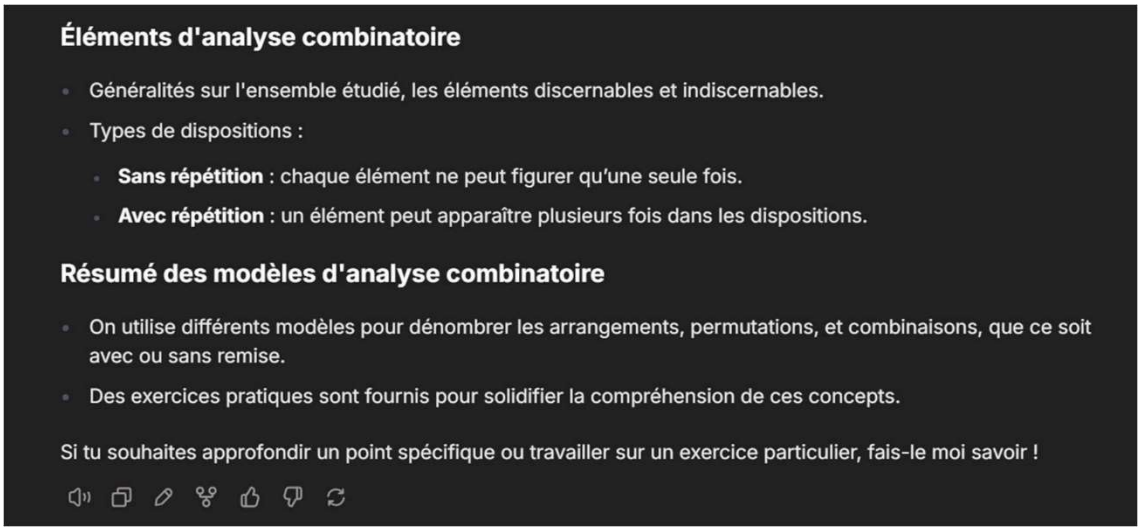
Figure 17 : (PAI) Analyse de la demande

Création personnelle (Bourinet, 2025)

Figure 18 : (PAI) Demande plus précise

Création personnelle (Bourinet, 2025)

Figure 19 : (PAI) Réponse finale



Création personnelle (Bourinet, 2025)

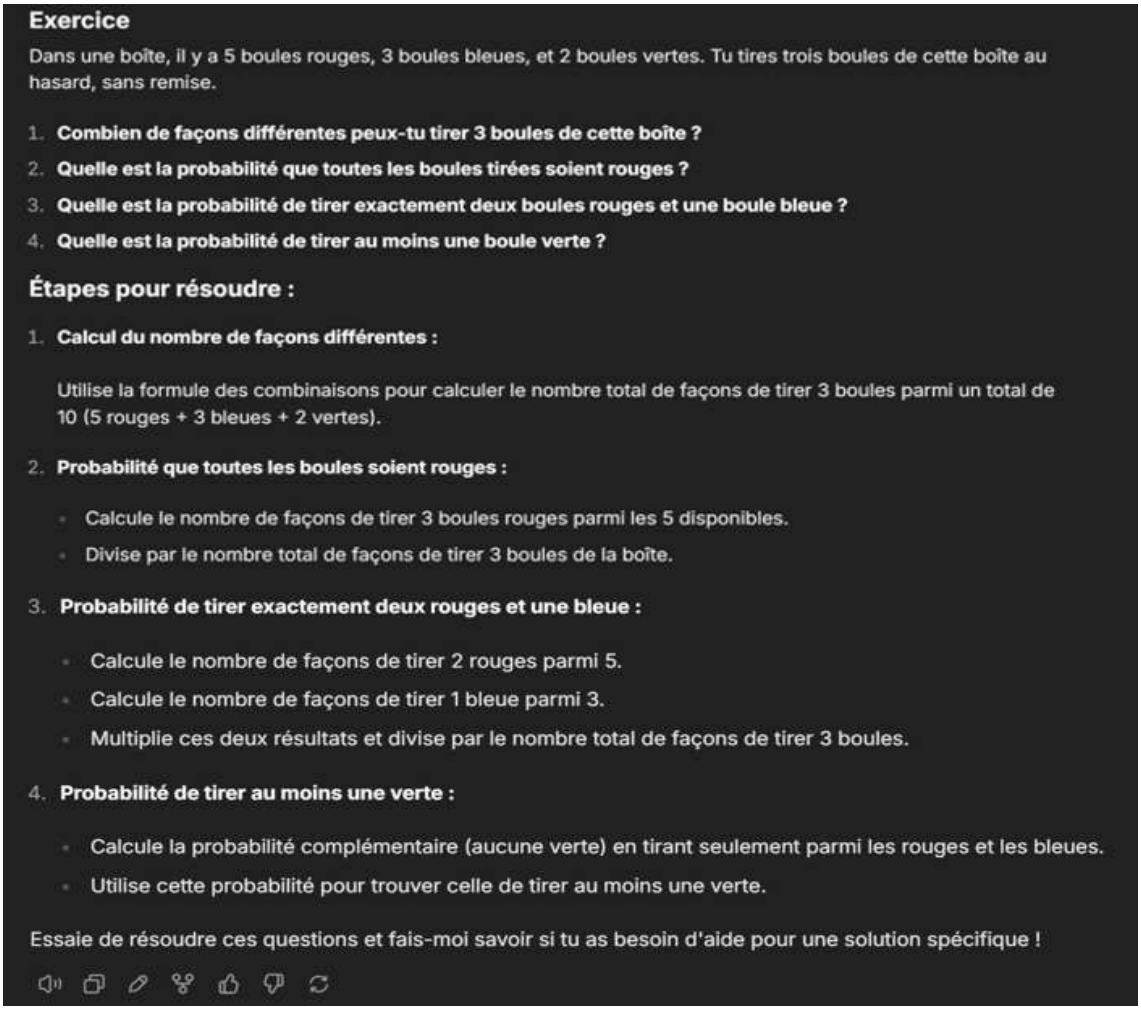
Figure 20 : (PAI) Indications concernant le professeur exercices

Création personnelle (Bourinet, 2025)

Figure 21 : (PAI) Call et run de l'agent en question

Création personnelle (Bourinet, 2025)

Figure 22 : (PAI) Réponse finale exercice



Exercice

Dans une boîte, il y a 5 boules rouges, 3 boules bleues, et 2 boules vertes. Tu tires trois boules de cette boîte au hasard, sans remise.

1. **Combien de façons différentes peux-tu tirer 3 boules de cette boîte ?**
2. **Quelle est la probabilité que toutes les boules tirées soient rouges ?**
3. **Quelle est la probabilité de tirer exactement deux boules rouges et une boule bleue ?**
4. **Quelle est la probabilité de tirer au moins une boule verte ?**

Étapes pour résoudre :

1. **Calcul du nombre de façons différentes :**

   Utilise la formule des combinaisons pour calculer le nombre total de façons de tirer 3 boules parmi un total de 10 (5 rouges + 3 bleues + 2 vertes).

2. **Probabilité que toutes les boules soient rouges :**
   - Calcule le nombre de façons de tirer 3 boules rouges parmi les 5 disponibles.
   - Divise par le nombre total de façons de tirer 3 boules de la boîte.

3. **Probabilité de tirer exactement deux rouges et une bleue :**
   - Calcule le nombre de façons de tirer 2 rouges parmi 5.
   - Calcule le nombre de façons de tirer 1 bleue parmi 3.
   - Multiplie ces deux résultats et divise par le nombre total de façons de tirer 3 boules.

4. **Probabilité de tirer au moins une verte :**
   - Calcule la probabilité complémentaire (aucune verte) en tirant seulement parmi les rouges et les bleues.
   - Utilise cette probabilité pour trouver celle de tirer au moins une verte.

Essaie de résoudre ces questions et fais-moi savoir si tu as besoin d'aide pour une solution spécifique !

Création personnelle (Bourinet, 2025)

7. Réponses problématique et questions de départ

Les problématiques soulevées dans ce travail ont guidé l'ensemble de la démarche, tant dans la structuration du rapport que dans le développement du prototype.

La distinction entre un simple modèle de langage (LLM) et un véritable agent IA a été clarifiée en démontrant que l'agent, au-delà de la génération de texte, s'inscrit dans une logique de rôle, d'action et de persistance.

Un LLM devient agent lorsqu'il est intégré dans une architecture capable d'interagir avec des outils, de conserver un historique, et d'exécuter des tâches orientées objectif.

L'orchestration multi-agents, quant à elle, a été abordée à travers l'étude de frameworks spécialisés (CrewAI, LangGraph, OpenAI SDK) et la conception d'un système fonctionnel mettant en scène plusieurs agents spécialisés autour d'un professeur IA. Cette orchestration a mis en lumière des enjeux clés tels que la coordination des flux, la gestion du contexte, et la délégation cohérente des tâches.

L'ajout d'un système de RAG a permis d'illustrer la manière dont l'accès à une base de connaissances externe enrichit les réponses produites par les agents, bien que son efficacité optimale dépende fortement de la qualité de l'indexation et du prompt engineering. Ainsi, ce travail propose une réponse complète aux enjeux initiaux, tout en reconnaissant que chacun de ces axes mériterait un approfondissement spécifique dans des recherches futures.

Conclusion

Ce travail de Bachelor représente avant tout un parcours stimulant et enrichissant. Je suis heureux d'avoir pu explorer un domaine qui m'était encore largement inconnu au début du projet, à l'exception de quelques bases en machine learning. En effet j'ai grandement apprécié découvrir tous les aspects autour des agents IA et surtout pouvoir d'avantage visualisé leur construction et objectifs. L'univers des agents s'est révélé bien plus vaste, complexe et passionnant que je ne l'avais imaginé.

Ce rapport m'a permis d'aborder des thématiques variées telles que la différenciation entre IA générative et agent, l'orchestration multi-agents, le rôle des outils comme le RAG ou le MCP, ou encore la gestion du contexte et de la mémoire dans les systèmes distribués. Chaque concept abordé a suscité un véritable intérêt, mais aussi une part de frustration : tous auraient mérité d'être explorés plus en profondeur. Il a été difficile de se renseigner sans se perdre dans toutes les directions et en faisant le tri parmi les informations. Mais la véritable frustration réside dans le fait que même après ce rapport je n'ai pas acquis le « concrètement » pour chaque point. J'ai souvent eu le sentiment de devoir poser des limites et de simplement rester en surface dans des domaines immenses, sans pouvoir m'y enfoncer autant que je l'aurais souhaité, faute de temps ou de recul technologique suffisant. La constante évolution du sujet rend difficile la stabilisation des savoirs.

Ce travail m'a néanmoins permis de visualiser concrètement la construction d'un agent, d'un système multi-agent, et d'un prototype fonctionnel incarné par le professeur IA personnalisé. Ce dernier constitue un exemple pratique, plus qu'un produit final abouti. Il démontre qu'un système d'agents peut être mis en œuvre avec les outils actuels. Il faut également garder en tête qu'évidement il n'est pas optimisé de mettre de l'IA absolument partout juste pour mettre de l'IA partout, je trouvais ça amusant de tester au maximum les possibilités via l'IA et d'avoir un maximum d'agent et de flux même si finalement cela rend la chose bien plus complexe qu'elle ne devrait l'être.

Ce projet s'inscrit davantage dans une logique exploratoire que dans une logique d'optimisation. Avec cette expérience, je reste conscient du chemin qu'il me reste à parcourir pour véritablement maîtriser ce domaine. Mon objectif, à travers ce rapport, était de proposer une vision globale et accessible des agents IA pour des personnes du domaine de l'IT, sans tomber dans la superficialité, mais sans non plus me perdre en voulant approfondir tous les points.

Bibliographie

FOREST, David, 2011. *Droit des données personnelles*. Paris : Gualino, 2011. Droit en action. ISBN 9782297015028.

FOREST, David, 2011. Droit des données personnelles. Paris : Gualino, 2011. Droit en action. ISBN 9782297015028.

ALMEIDA, Paulo Sérgio, 2024. A Framework for Consistency Models in Distributed Systems. arXiv:2411.16355. arXiv. arXiv:2411.16355. DOI 10.48550/arXiv.2411.16355. arXiv:2411.16355 [cs]

ANTHROPIC, 2024. Introducing the Model Context Protocol. [en ligne]. 2024. Disponible à l'adresse : https://www.anthropic.com/news/model-context-protocol [consulté le 18 juillet 2025].

ANTHROPIC, 2025. How we built our multi-agent research system. [en ligne]. 2025. Disponible à l'adresse : https://www.anthropic.com/engineering/built-multi-agent-research-system [consulté le 22 juillet 2025].

API Reference - OpenAI API, [en ligne]. 2025. Disponible à l'adresse : https://platform.openai.com [consulté le 12 juin 2025].

AutoGPT Documentation, [en ligne]. Disponible à l'adresse : https://docs.agpt.co/ [consulté le 04 mai 2025].

BLANC, Herve, 2024. RAG Génération Augmentée par Récupération d'informations. biZNov [en ligne]. 14 février 2024. Disponible à l'adresse : https://www.biznov.fr/post/rag_génération_augmentée_par_récupération [consulté le 2 juillet 2025].

BOMMASANI, Rishi et al., 2022. On the Opportunities and Risks of Foundation Models. arXiv:2108.07258. arXiv. arXiv:2108.07258. DOI 10.48550/arXiv.2108.07258. arXiv:2108.07258 [cs]

CHEN, Yu-Zhen Janice et al., 2023. On-Demand Communication for Asynchronous Multi-Agent Bandits. arXiv:2302.07446. arXiv. arXiv:2302.07446. DOI 10.48550/arXiv.2302.07446. arXiv:2302.07446 [cs]

CLAYTONSIEMENS. Azure Architecture Center - Azure Architecture Center. [en ligne]. Disponible à l'adresse : https://learn.microsoft.com/en-us/azure/architecture/ [consulté le 29 juillet 2025].

COLELOUGH, Brandon C. et REGLI, William, 2025. Neuro-Symbolic AI in 2024: A Systematic Review. arXiv:2501.05435. arXiv. arXiv:2501.05435. DOI 10.48550/arXiv.2501.05435. arXiv:2501.05435 [cs]

CORTES, Corinna et VAPNIK, Vladimir, 1995. Support-vector networks. Machine Learning. Vol. 20, no 3, pp. 273-297. DOI 10.1007/BF00994018.

CQRS pattern - AWS Prescriptive Guidance, [en ligne]. Disponible à l'adresse : https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/cqrs-pattern.html [consulté le 23 juin 2025].

crewAIInc/crewAI [logiciel] [en ligne]. 30 mars 2025. crewAI. [consulté le 30 mars 2025]. Disponible à l'adresse : https://github.com/crewAIInc/crewAI [consulté le 30 mars 2025].

Cursor – Welcome, Cursor [en ligne]. Disponible à l'adresse : https://docs.cursor.com/en/welcome [consulté le 2 avril 2025].

DARAGHMI, Eman, ZHANG, Cheng-Pu et YUAN, Shyan-Ming, 2022. Enhancing Saga Pattern for Distributed Transactions within a Microservices Architecture. Applied Sciences. Vol. 12, no 12, p. 6242. DOI 10.3390/app12126242.

deepseek-r1:7b, [en ligne]. Disponible à l'adresse : https://ollama.com/deepseek-r1:7b [consulté le 29 juillet 2025].

ERIC-URBAN, 2025. Azure AI Foundry documentation. [en ligne]. 2025. Disponible à l'adresse : https://learn.microsoft.com/en-us/azure/ai-foundry/ [consulté le 12 juillet 2025].

Explore n8n Docs: Your Resource for Workflow Automation and Integrations | n8n Docs, [en ligne]. Disponible à l'adresse : https://docs.n8n.io/ [consulté le 26 mars 2025].

FAN, Linxi et al., 2022. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge. arXiv:2206.08853. arXiv. arXiv:2206.08853. DOI 10.48550/arXiv.2206.08853. arXiv:2206.08853 [cs]

FIRAT, Mehmet et KULELİ, Saniye, 2024. What if GPT4 Became Autonomous: The Auto-GPT Project and Use Cases. Journal of Emerging Computer Technologies. Vol. 3, no 1, pp. 1-6. DOI 10.57020/ject.1297961.

GAO, Yunfan et al., 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997. arXiv. arXiv:2312.10997. DOI 10.48550/arXiv.2312.10997. arXiv:2312.10997 [cs]

Get Started, 2025 [en ligne]. Disponible à l'adresse : https://librechat.ai/docs [consulté le 20 juin 2025].

GUO, Taicheng et al., 2024. Large Language Model based Multi-Agents: A Survey of Progress and Challenges. arXiv:2402.01680. arXiv. arXiv:2402.01680. DOI 10.48550/arXiv.2402.01680. arXiv:2402.01680 [cs]

HOU, Xinyi et al., 2025. Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. arXiv:2503.23278. arXiv. arXiv:2503.23278. DOI 10.48550/arXiv.2503.23278. arXiv:2503.23278 [cs]

HU, Guanzhou, ARPACI-DUSSEAU, Andrea et ARPACI-DUSSEAU, Remzi, 2025. A

Unified, Practical, and Understandable Model of Non-transactional Consistency Levels in Distributed Replication. arXiv:2409.01576. arXiv. arXiv:2409.01576. DOI 10.48550/arXiv.2409.01576. arXiv:2409.01576 [cs]

HUGHES, Alyssa, 2023. AutoGen: Enabling next-generation large language model applications. Microsoft Research [en ligne]. 25 septembre 2023. Disponible à l'adresse : https://www.microsoft.com/en-us/research/blog/autogen-enabling-next-generation-large-language-model-applications/ [consulté le 23 juin 2025].

Introduction | LangChain, [en ligne]. Disponible à l'adresse :

https://python.langchain.com/docs/introduction/ [consulté le 15 avril 2025].

IUSZTIN, Paul, 2025. AI Agent Memory: Short/Long Term, RAG, Agentic RAG. Decoding ML [en ligne]. 17 avril 2025. Disponible à l'adresse : https://decodingml.substack.com/p/memory-the-secret-sauce-of-ai-agents [consulté le 28 juin 2025].

LANGGRAPH, 2025. Project: Building Ambient Agents with LangGraph. LangChain Academy [en ligne]. 2025. Disponible à l'adresse : https://academy.langchain.com/courses/ambient-agents [consulté le 21 juillet 2025].

LEWIS, Patrick et al., 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401. arXiv. arXiv:2005.11401. DOI 10.48550/arXiv.2005.11401. arXiv:2005.11401 [cs]

MALKOV, Yu A. et YASHUNIN, D. A., 2018. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. arXiv:1603.09320. arXiv. arXiv:1603.09320. DOI 10.48550/arXiv.1603.09320. arXiv:1603.09320 [cs]

MCCARTHY, John, 1959. PROGRAMS WITH COMMON SENSE. .

MCP, 2025. Specification. Model Context Protocol [en ligne]. 2025. Disponible à l'adresse : https://modelcontextprotocol.io/specification/2025-06-18 [consulté le 17 juillet 2025].

MEROLA, Carlo et SINGH, Jaspinder, 2025. Reconstructing Context: Evaluating Advanced Chunking Strategies for Retrieval-Augmented Generation. arXiv:2504.19754. arXiv. arXiv:2504.19754. DOI 10.48550/arXiv.2504.19754. arXiv:2504.19754 [cs]

microsoft/ai-agents-for-beginners [logiciel] [en ligne]. Microsoft. [consulté le 2 juillet 2025]. Disponible à l'adresse : https://github.com/microsoft/ai-agents-for- beginners [consulté le 2 juillet 2025].

microsoft/autogen [logiciel] [en ligne]. Microsoft. [consulté le 19 mai 2025]. Disponible à l'adresse : https://github.com/microsoft/autogen [consulté le 19 mai 2025].

microsoft/generative-ai-for-beginners [logiciel] [en ligne]. 5 juin 2025. Microsoft. [consulté le 5 juin 2025]. Disponible à l'adresse : https://github.com/microsoft/generative-ai-for-beginners [consulté le 5 juin 2025].

Ollama, [en ligne]. Disponible à l'adresse : https://ollama.com [consulté le 9 mars 2025].

Pydantic AI, [en ligne]. Disponible à l'adresse : https://ai.pydantic.dev/ [consulté le 6 juillet 2025].

SAPKOTA, Ranjan, ROUMELIOTIS, Konstantinos I. et KARKEE, Manoj, 2025. AI Agents vs. Agentic AI: A Conceptual Taxonomy, Applications and Challenge. arXiv:2505.10468. arXiv. arXiv:2505.10468. DOI 10.48550/arXiv.2505.10468. arXiv:2505.10468 [cs]

SHAHEEN, Abdelrhman et al., 2025. Reinforcement Learning in Strategy-Based and Atari Games: A Review of Google DeepMinds Innovations. arXiv:2502.10303. arXiv. arXiv:2502.10303. DOI 10.48550/arXiv.2502.10303. arXiv:2502.10303 [cs]

SHU, Raphael et al., 2024. Towards Effective GenAI Multi-Agent Collaboration: Design and Evaluation for Enterprise Applications. arXiv:2412.05449. arXiv. arXiv:2412.05449. DOI 10.48550/arXiv.2412.05449. arXiv:2412.05449 [cs]

SILVER, David et al., 2016. Mastering the game of Go with deep neural networks and tree search. Nature. Vol. 529, no 7587, pp. 484-489. DOI 10.1038/nature16961.

SIMMERING, Paul, 2024. Type-safe LLM agents with PydanticAI – Paul Simmering. [en ligne]. 16 décembre 2024. Disponible à l'adresse : https://simmering.dev/blog/pydantic-ai/ [consulté le 7 juillet 2025].

SMITH, Reid G, 1980. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. IEEE TRANSACTIONS ON COMPUTERS. No 12.

STEWART, Ryan, PALMER, Todd S et BAYS, Samuel, 2022. An agent-based blackboard system for multi-objective optimization. Journal of Computational Design and Engineering. Vol. 9, no 2, pp. 480-506. DOI 10.1093/jcde/qwac009.

TALLAM, Krti, 2025. From Autonomous Agents to Integrated Systems, A New Paradigm: Orchestrated Distributed Intelligence. arXiv:2503.13754. arXiv. arXiv:2503.13754. DOI 10.48550/arXiv.2503.13754. arXiv:2503.13754 [eess]

TAMPUU, Ardi et al., 2017. Multiagent cooperation and competition with deep reinforcement learning. PLOS ONE. Vol. 12, no 4, p. e0172395. DOI 10.1371/journal.pone.0172395.

WANG, Lei et al., 2024. A Survey on Large Language Model based Autonomous Agents. Frontiers of Computer Science. Vol. 18, no 6. DOI 10.1007/s11704-024-40231-1. arXiv:2308.11432 [cs]

WANG, Xuezhi et ZHOU, Denny, 2024. Chain-of-Thought Reasoning Without Prompting. arXiv:2402.10200. arXiv. arXiv:2402.10200. DOI 10.48550/arXiv.2402.10200. arXiv:2402.10200 [cs]

WANG, Yuqing et ZHAO, Yun, 2024. Metacognitive Prompting Improves Understanding in Large Language Models. arXiv:2308.05342. arXiv. arXiv:2308.05342. DOI 10.48550/arXiv.2308.05342. arXiv:2308.05342 [cs]

WEI, Jason et al., 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903. arXiv. arXiv:2201.11903. DOI 10.48550/arXiv.2201.11903. arXiv:2201.11903 [cs]

WU, Shengqiong et al., 2025. Towards Semantic Equivalence of Tokenization in Multimodal LLM. arXiv:2406.05127. arXiv. arXiv:2406.05127. DOI 10.48550/arXiv.2406.05127. arXiv:2406.05127 [cs]

XI, Zhiheng et al., 2023. The Rise and Potential of Large Language Model Based Agents: A Survey. arXiv:2309.07864. arXiv. arXiv:2309.07864. DOI 10.48550/arXiv.2309.07864. arXiv:2309.07864 [cs]

YAN, Bingyu et al., 2025. Beyond Self-Talk: A Communication-Centric Survey of LLM-Based Multi-Agent Systems. arXiv:2502.14321. arXiv. arXiv:2502.14321. DOI 10.48550/arXiv.2502.14321. arXiv:2502.14321 [cs]

YAO, Shunyu et al., 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629. arXiv. arXiv:2210.03629. DOI 10.48550/arXiv.2210.03629. arXiv:2210.03629 [cs]

YE, Qinyuan et al., 2024. Prompt Engineering a Prompt Engineer. arXiv:2311.05661. arXiv. arXiv:2311.05661. DOI 10.48550/arXiv.2311.05661. arXiv:2311.05661 [cs]

ZHANG, Guibin et al., 2024. G-Designer: Architecting Multi-agent Communication Topologies via Graph Neural Networks. arXiv:2410.11782. arXiv. arXiv:2410.11782. DOI 10.48550/arXiv.2410.11782. arXiv:2410.11782 [cs]

ZHAO, Penghao et al., 2024. Retrieval-Augmented Generation for AI-Generated Content: A Survey. arXiv:2402.19473. arXiv. arXiv:2402.19473. DOI 10.48550/arXiv.2402.19473. arXiv:2402.19473 [cs]

ZHAO, Siyun et al., 2024. Retrieval Augmented Generation (RAG) and Beyond: A Comprehensive Survey on How to Make your LLMs use External Data More Wisely. arXiv:2409.14924. arXiv. arXiv:2409.14924. DOI 10.48550/arXiv.2409.14924. arXiv:2409.14924 [cs]